

Optimal Trajectory Generation and Learning Control for Robot Table Tennis

Optimale Trajektoriengenerierung und Lernkontrolle für Roboter-Tischtennis

Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation von Okan Koç aus Istanbul

Oktober 2019 — Darmstadt — D 17



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Optimal Trajectory Generation and Learning Control for Robot Table Tennis
Optimale Trajektoriengenerierung und Lernkontrolle für Roboter-Tischtennis

Genehmigte Dissertation von Okan Koç aus Istanbul

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Sethu Vijayakumar

Tag der Einreichung: 3 April 2019

Tag der mündlichen Prüfung: 24 Oktober 2018

Darmstadt — D 17

Please cite this document with:

URN: urn:nbn:de:tuba-tuprints-89486

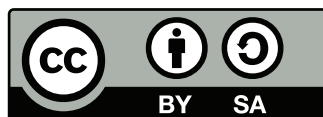
URL: <http://tuprints.ulb.tu-darmstadt.de/id/eprint/8948>

Dieses Dokument wird bereitgestellt von tuprints,

E-Publishing-Service der TU Darmstadt

<http://tuprints.ulb.tu-darmstadt.de>

tuprints@ulb.tu-darmstadt.de



This publication is licensed under the following Creative Commons License:

Attribution – ShareAlike 4.0 International

<http://creativecommons.org/licenses/by-sa/4.0/>

For my friends and my family

Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 6. Oktober 2019

(Okan Koç)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, October 6, 2019

(Okan Koç)

Abstract

As robots become more capable in terms of hardware, and more complex tasks are considered, optimality starts playing a more important role in the design of algorithms implemented in these systems. Optimality is a guiding principle that directs the computation of feasible and efficient solutions to different robotics tasks. In control theory, this principle is implemented online as a set of efficient numerical optimization algorithms, that in addition to solving the task, purports to save a suitably defined effort or energy term.

This thesis investigates trajectory generation, learning and control for dynamic tasks from the unifying point of view of optimization. As an application, we focus on Table Tennis, a challenging task where robots are yet to outperform humans. We believe that the required dexterity and accuracy for this dynamical task hinges on the developments in online optimization and efficient learning algorithms.

We consider trajectory generation for table tennis in the first part of the thesis. In highly dynamic tasks like table tennis that involve moving targets, planning is necessary to figure out when, where and how to intercept the target. Motion planning can be very challenging in robotic table tennis in particular, due to time constraints, dimension of the search space and joint limits. Conventional planning algorithms often rely on a fixed virtual hitting plane to construct robot striking trajectories. These algorithms, however, generate restrictive strokes and can result in unnatural strategies when compared with human playing. In this thesis, we introduce a new trajectory generation framework for robotic table tennis that does not involve a fixed hitting plane. A free-time optimal control approach is used to derive two different trajectory optimizers. The resulting two algorithms, Focused Player and Defensive Player, encode two different play-styles. We evaluate their performance in simulation and in our robot table tennis platform with a high speed cable-driven seven DOF robot arm. The algorithms return the balls with a higher probability to the opponent's court when compared with a virtual hitting plane based method. Moreover, both can be run online and the trajectories can be corrected with new ball observations.

In the second part of the thesis, we look at how such trajectories, computed on the kinematics level, can be tracked accurately with learning control based approaches. Highly dynamic tasks like table tennis require large accelerations and precise tracking for successful performance. To track desired trajectories well, such tasks usually rely on accurate models and/or high gain feedback. While kinematic optimization allows for efficient representation and online generation of hitting trajectories, learning to track such dynamic movements with inaccurate models remains an open problem. In particular, stability issues surrounding the learning performance, in the iteration domain, can prevent the successful implementation of model based learning approaches. To achieve accurate tracking for these tasks in a stable and efficient way, we propose a new adaptive Iterative Learning Control algorithm that is implemented efficiently using a recursive approach. Moreover, covariance estimates of model matrices are used to exercise caution during learning. We evaluate the performance of the proposed approach in our robotic table tennis platform, where we show how the performance of two Barrett WAMs can

be optimized. Our implementation on the table tennis platform compares favorably with two state-of-the-art approaches.

Finally, we discuss an alternative learning from demonstrations approach, where we learn sparse representations from demonstrated movements. Learning from demonstrations is an easy and intuitive way to show examples of successful behavior to a robot. However, the fact that humans optimize or take advantage of their body and not of the robot, usually called the embodiment problem in robotics, often prevents industrial robots from executing the task in a straightforward way. The shown movements often do not or cannot utilize the degrees of freedom of the robot efficiently, and typically suffer from excessive execution errors. In the last chapter, we show a new approach that can alleviate some of these difficulties by learning sparse representations of movement. Moreover, the number of learned parameters are independent of the degrees of freedom of the robot. Sparsity is a desirable feature for policy search Reinforcement Learning algorithms that adapt the parameters of these movement primitives. By ranking the learned parameters on the Elastic Net path in terms of importance, we note that our approach could be potentially useful to combat the curse of dimensionality in robot learning applications. We show preliminary results on the real robot setup, including a successful table tennis serve using our new movement primitive representation.

Throughout the thesis, we present and analyze in detail new control and learning algorithms. Efficient online optimization approaches are presented that can be used to solve not just table tennis problems, but they can be adapted to solve different dynamic tasks.

Zusammenfassung

Roboter werden in Bezug auf Ihre Hardware immer leistungsfähiger und können somit für komplexere Aufgaben in Betracht gezogen werden. Die Optimalität von Algorithmen die für diese Systeme implementiert werden, spielt hierbei eine immer wichtigere Rolle und ist ein Leitprinzip, das die Berechnung praktikabler und effizienter Lösungen für verschiedene Aufgaben in der Robotik steuert. In der Kontrolltheorie wird dieses Prinzip unter der Vorgabe eines geeignet definierten Aufwands durch effiziente numerische Optimierungsalgorithmen umgesetzt.

Diese Arbeit untersucht die Erzeugung, das Lernen und die Kontrolle von Trajektorien für dynamische Aufgaben, ausgehend vom vereinigenden Standpunkt der Optimierung. Als Anwendung konzentrieren wir uns auf das Tischtennis, eine anspruchsvolle Aufgabe bei der Roboter den Menschen noch nicht übertreffen. Wir glauben, dass die erforderliche Geschicklichkeit und Genauigkeit für diese dynamische Aufgabe von den Entwicklungen der Online-Optimierung und effizienten Lernalgorithmen abhängt.

Wir betrachten die Erzeugung von Trajektorien für das Tischtennis im ersten Teil der Arbeit. Bei hochdynamischen Aufgaben wie Tischtennis, bei denen Ziele bewegt werden, ist Planung notwendig um herauszufinden, wann, wo und wie man das Ziel treffen soll. Die Planung von Bewegungen kann insbesondere im Roboter-Tischtennis aufgrund von Zeitbeschränkungen, Dimension des Suchraums und Gelenkgrenzen sehr schwierig sein. Herkömmliche Algorithmen für die Planung von Schlagbewegungen beruhen häufig auf einer festen virtuellen Trefferebene. Damit können jedoch nur restriktive Schläge erzeugt werden und dies kann im Vergleich zum menschlichen Spiel zu unnatürlichen Strategien führen. Deshalb stellen wir in dieser Arbeit ein

neues Framework zur Erzeugung von Trajektorien für das Roboter-Tischtennis vor, welches keine feste Trefferfläche beinhaltet. Hierfür wird ein optimaler freier Steuerungsansatz verwendet, um zwei verschiedene Optimierungsansätze für die Trajektorien abzuleiten. Die resultierenden zwei Algorithmen, Focused Player und Defensive Player, kodieren jeweils zwei verschiedene Spielstile. Wir evaluieren ihre Leistung in der Simulation und mit Hilfe unserer Plattform für Roboter-Tischtennis, bestehend aus einem Kabel betriebenen Hochgeschwindigkeits-Arm mit sieben Freiheitsgraden. Im Vergleich zu Methoden die auf einer virtuellen Trefferebene basieren, geben beide Algorithmen die Bälle mit einer höheren Wahrscheinlichkeit in die Hälfte des Gegners zurück. Darüber hinaus können beide Ansätze online ausgeführt werden und die Trajektorien können mit neuen Ballbeobachtungen korrigiert werden.

Im zweiten Teil der Arbeit betrachten wir, wie Trajektorien die auf der Ebene der Kinematik berechnet werden, mit Lernbasierten Ansätzen genau verfolgt werden können. Hochdynamische Aufgaben wie das Tischtennis erfordern große Beschleunigungen und ein präzises Tracking für eine erfolgreiche Leistung. Um die gewünschten Trajektorien gut zu verfolgen, benötigen solche Aufgaben in der Regel genaue Modelle und/oder Feedback mit hohem Signal. Während die kinematische Optimierung eine effiziente Darstellung und Online-Erzeugung von Schlagbahnen ermöglicht, ist das Lernen der Verfolgung von dynamischen Bewegungen mit ungenauen Modellen, weiterhin ein offenes Problem. Insbesondere können Stabilitätsprobleme von iterativen Verfahren im Zusammenhang mit der Lernleistung, die erfolgreiche Implementierung von modellbasierten Lernansätzen verhindern. Um eine genaue Ausführung für diese Aufgaben auf eine stabile und effiziente Weise zu erreichen, schlagen wir einen neuen adaptiv iterativen Lernsteuerungsansatz vor, der mit Hilfe eines rekursiven Ansatzes effizient implementiert wird. Darüber hinaus werden Kovarianzmatrizen der Modelle verwendet, um Unsicherheitsschätzungen in den Lernprozess mit einzubeziehen. Wir bewerten die Leistung des vorgeschlagenen Ansatzes an Hand unserer Plattform für Roboter-Tischtennis, indem wir zeigen wie der Arm optimiert werden kann. Im Vergleich zum aktuellsten Stand der Technik weist unsere Umsetzung Vorteile auf.

Schließlich diskutieren wir einen alternativen Ansatz zum Lernen von dünnbesetzten Darstellungen aus demonstrierten Bewegungen. Lernen aus Demonstrationen ist eine einfache und intuitive Möglichkeit, einem Roboter Beispiele für erfolgreiches Verhalten zu zeigen. Die Tatsache, dass Menschen ihren Körper und nicht den Roboter optimieren oder nutzen, wird in der Robotik oft als Problem der Verkörperung bezeichnet und verhindert, dass Industrieroboter die gleiche Aufgabe einfach ausführen können. Die gezeigten Bewegungen können die Freiheitsgrade des Roboters oft nicht effizient nutzen und leiden typischerweise unter exzessiven Ausführungsfehlern. Im letzten Kapitel zeigen wir einen neuen Ansatz, der einige dieser Schwierigkeiten lindern kann, indem dünn besetzte Darstellungen von Bewegungen gelernt werden. Darüber hinaus ist die Anzahl der gelernten Parameter unabhängig von den Freiheitsgraden des Roboters. Eine dünn besetzte Darstellung ist ein wünschenswertes Merkmal für die Suche nach Strategien die im Bestärkenden Lernen versuchen die Parameter von Bewegungsprimitiven anzupassen. Indem wir die erlernten Parameter auf dem Pfad eines elastischen Netzes nach Wichtigkeit ordnen, stellen wir fest, dass unser Ansatz möglicherweise den Fluch der Dimensionalität für Lernanwendungen in der Robotik mildern kann. Wir zeigen vorläufige Ergebnisse auf unserer physischen Roboterplattform, einschließlich eines erfolgreichen Tischtennis-Aufschlags unter Verwendung unserer neuen Darstellung von Bewegungsprimitiven.

Im Verlauf der Arbeit präsentieren und analysieren wir neue Steuerungs- und Lernalgorithmen. Es werden Online- und effiziente Optimierungsansätze vorgestellt, mit denen nicht nur

Probleme im Tischtennis, sondern auch andere dynamische Aufgaben gelöst werden können.

Acknowledgments

I would like to thank

- Jan Peters for his friendliness, his guidance and feedback throughout the PhD.
- Bernhard Schölkopf for creating such an amazing environment at MPI with intelligent, lovely people and great scientific discussions. I really enjoyed the academic freedom Jan and Bernhard gave me to pursue my interests at will.
- My teammates Dieter, Sebastian and Diego for the table-tennis matches and for the good times we spent together.
- My colleagues at MPI, who have made my stay in Germany very pleasant.
- Guilherme Maeda for reading my papers and giving me good feedback.
- Yanlong Huang for his kindness and friendship. But I am much better than you in table tennis!
- My family for investing so much time and effort on me!
- Our nice coffee machines at MPI :)
- Finally, last but not least, my girlfriend Cristina for her constant friendship, love and support.

Contents

1. Introduction	2
1.1. Motivation	2
1.2. Contributions	3
1.3. Outline	4
2. Online Optimal Trajectory Generation for Robot Table Tennis	6
2.1. Introduction	6
2.2. Related Work	7
2.3. Problem Statement	8
2.4. The Focused Player	13
2.5. The Defensive Player	18
2.6. Experiments & Evaluations	24
2.7. Conclusion	34
3. Optimizing Execution of Robot Striking Movements with Learning Control	36
3.1. Introduction	36
3.2. Problem Statement and Background	40
3.3. Model Adaptation	42
3.4. Cautious Learning Control	45
3.5. Online Implementation for Robot Table Tennis	47
3.6. Evaluations and Experiments	50
3.7. Conclusion and Future Work	60
4. Learning to Serve: an Experimental Study	62
4.1. Introduction	62
4.2. Notation	64
4.3. Method	65
4.4. Experiments	69
4.5. Conclusion	73
5. Conclusion	75
5.1. Summary	75
5.2. Open Problems	76
Bibliography	79
A. Appendix	85
A.1. Parameter Estimation	85
A.2. Robust Kalman Filtering	87

A.3. Derivations for Chapter 2	87
A.4. Derivations for Chapter 3	91
A.5. Papers Reviewed	94

Figures and Tables

List of Figures

- 2.1. Robotic table tennis setup with four cameras on the corners of the ceiling tracking the ball at 60 Hz. We present two optimal control based trajectory generation algorithms that encode defensive and goal-oriented styles of playing. A constrained nonlinear optimization problem is solved in both cases to find an optimal striking trajectory as well as an optimal striking time. 7
- 2.2. Fixing a virtual hitting plane (VHP) can make the generated trajectories unnecessarily restrictive and the resulting inverse kinematics may be infeasible. Instead the whole ball trajectory should be considered in a trajectory generation framework and the hitting time as well as the hitting point should be optimized. The predicted ball trajectory and a feasible racket trajectory are shown in red and black, respectively. VHP is shown as a dotted gray line, the workspace of the robot is shown as an ellipsoidal light blue region. 8
- 2.3. In table tennis, robot trajectories (blue) can be seen as reactions to predicted ball trajectories. The players are free to decide where, when and how to intercept the ball. However, the resulting outgoing ball trajectories (orange) need to be *feasible*: the ball has to pass above the net and land on the opponent's court. The feasible region above the net is drawn in transparent green. The rules of the game can be captured as constraints for generating robot striking trajectories. . . 10
- 2.4. Ball prediction schema for table tennis. After estimating the initial ball position, velocity and spin, the future path of the ball can be predicted using the flight model, the rebound model and the racket-ball contact model. The trajectory generation framework uses these models to compute desired striking trajectories. 10
- 2.5. Graphical representation of table tennis interactions. The hybrid system for the table tennis ball is described by the flight dynamics, governed by a set of differential equations, as well as a discrete hitting event \mathcal{H} that changes the ball velocity from $\dot{\mathbf{b}}(T^-)$ to $\dot{\mathbf{b}}(T^+)$ at the hitting time T . The control variables for the reduced optimization problem are located in the light blue rectangle. Racket constraints that are enforced by *Focused Player* to land the ball to a fixed location are indicated in the red rectangle. *Defensive Player* on the other hand, directly enforces the task (landing and net) constraints, located in the orange rectangle, without additional constraints. By additionally checking for the hitting condition \mathcal{H} in the optimization, this problem can be cast as a (standard) continuous optimal control problem, where the decision variables \mathbf{q}_f , $\dot{\mathbf{q}}_f$ and T continuously affect the outgoing ball velocity, the ball net and landing positions, through the repeated application of the flight model (2.9) and the contact model (2.14). . . . 21

2.6.	For simulating the performance of the virtual hitting plane (VHP) based method in a fair way, the results are averaged over four different VHP locations. The first and third plane locations are shown in the figure. Out of 50 balls each, the VHP at $y = -0.7$, $y = -0.6$, $y = -0.5$, $y = -0.4$ return 31, 37, 28, 29 balls respectively.	25
2.7.	As an alternative to computing the trajectory parameters online, [1] proposed a lookup table to generate trajectories. Performance of the trajectory generation framework using a lookup table is shown in blue. Results are averaged over 5 different runs. As the number of stored lookup table samples increase, the performance approaches that of the online trajectory generation in (a). However, as shown in (b), even the performance of a lookup table with 4000 entries degrades quickly whenever ball position and velocity estimates are not close to the values stored in the lookup table.	26
2.8.	Histogram of the runtime distributions of the two players, evaluated over 500 random test instances. Both algorithms FP and DP have an average runtime of about 25 ms, but for FP, the distribution is wider. For evaluating DP we have regressed on a lookup table using k-nearest-neighbor (kNN) regression. Without kNN, the runtime distribution for DP concentrates sharply around 50 ms.	27
2.9.	Simulation results comparing the return accuracy of three table tennis players. In (a), ball positions are observed with Gaussian white noise. In (b), there is an additional mismatch due to unknown topspin. Out of 200 balls, 14 and 12 incoming balls did not bounce legally and were not considered for trajectory generation, respectively. The other balls that were not counted as returns were either missed, or did not land legally on the opponent's court.	28
2.10.	Mean squared prediction error (red curve) is reduced as more balls are observed. The ball observations are used until contact with racket occurs and the results are averaged over 100 different real ball trials. Correcting for ball prediction error is critical for a robust table tennis performance, as the balls typically come with a high spin. Balls seem to lose some spin after rebound and the prediction error decreases faster. In this case this phenomenon can be observed after about 25 ball observations, where the change in the average slope of the red curve can be seen.	31
2.11.	Summary of real robot table tennis experiment results comparing three table tennis players. Bar plot values show the successful return % averaged over different starting postures and initial ball positions. The error bars indicate the standard deviation over a total of 200 trial runs.	32
2.12.	Overall, FP is able to return about 40 – 60% of the balls to the opponent's court. Setting the desired landing position on the right side of the table, with a desired landing time of $T_{\text{land}} = 0.4$ seconds, leads to the best performance ($\sim 60\%$) in our table tennis setup. Some example landing locations are indicated in orange in (b). Setting the desired landing position closer to the center of the opponent's court decreases the accuracy down to 40 – 50%, increasing also the variance of the landing locations, as shown in (a). DP in (c) with a landing accuracy of 80% has the highest variance in terms of the ball landing locations, as its returning criterion considers the whole opponent's court.	33

2.13.	Two example table tennis trials recorded in the table tennis setup are shown on the left hand side. The top two screenshots show the <i>Focused Player</i> (FP) in action, and the bottom four the <i>Defensive Player</i> (DP). Unlike FP, DP does not bring the robot back to the same initial posture (screenshots 3 vs. 6). Successful strike and the valid landing on the opponent's court for DP can be seen in the screenshots 4 – 5. Balls are highlighted with green dashed circles for visibility. The plot in the upper right figure shows the recordings from the cameras and the robot sensors, corresponding to the hitting movement in screenshots 1 and 2. The blue dots are the ball observations coming from cameras 3 and 4. The desired Cartesian trajectory is drawn in red, and the actual trajectory, in black.	33
2.14.	Four consecutive lands are shown for the <i>Defensive Player</i> (DP). In each trial, the arm goes back to a different resting posture.	34
2.15.	Tracking errors are shown for each joint. The desired joint positions and velocities are tracked with a PD controller. The deviation from the desired hitting point, shown as an orange dot in Figure 2.13, was for this example within three centimeters of the racket center, resulting in a hit.	35
3.1.	Our robot table tennis platform where a seven degree of freedom Barrett WAM arm is shown facing a ball-launcher. The ball is tracked using four cameras on the ceiling. Whenever a ball is approaching the robot, reference trajectories are computed online in order to return the ball to a desired location on the opponent's court. Such trajectories can be optimized on the kinematics level [2], however it is hard to execute them accurately without having access to accurate dynamics models. Iterative Learning Control, using inaccurate models, can still lead to an efficient approach for learning to track these trajectories.	36
3.2.	Learning performance of ILC, using inaccurate models without incorporating a notion of uncertainty, may not be monotonic in practice. One can observe ripples that move through the trajectory which can cause instability or damage the robot. In simulations we can create this effect easily by increasing the spectral norm of the difference between the nominal and the actual (lifted) dynamics matrices. The desired trajectory for the first state \mathbf{x}_1 is shown in dashed red on the left-hand side for a two dimensional linear time invariant system. The second plot shows the ILC feedforward commands for this particular trajectory and state. The third plot shows the Frobenius norm of the trajectory deviations, \mathcal{J}_k , plotted over the iterations k . The nonmonotonicity of the learning performance is aggravated, as the mismatch scale α controlling the spectral norm of the difference is increased. Increasing α further can prevent even asymptotic stability. The curves were generated by direct inversion of the (lifted) model. Our proposed Bayesian approach, on the other hand, minimizing the expected cost throughout the iterations, uses the posterior over the dynamics model parameters to make more cautious decisions.	38

- 3.3. Broyden's method [3], which can be considered as an adaptation framework within ILC, is a limiting case of Linear Bayesian Regression (LBR). As the forgetting factor λ of an exponentially weighted LBR model goes to zero, LBR transitions to Broyden's method. Broyden's method is very sensitive to noise and adapts very aggressively. Throughout the chapter, we discuss and evaluate several adaptation laws, that are less sensitive to noise but are still flexible. The Figure shows the evolution of the identification error norm for an unknown linear time-varying system. The Frobenius norm of the difference between the adapted model matrices ($\mathbf{A}_{k,j}$ and $\mathbf{B}_{k,j}$) and the actual (fixed) matrices (denoted as identification error norm) are plotted for each iteration $k = 1, \dots, 50$ 44
- 3.4. ILC in recursive form is evaluated on random linear time-varying (LTV) systems. The Frobenius norm of the trajectory deviations, \mathcal{J}_k , is plotted over the iterations k . Results are averaged over ten experiments, where for each experiment, trajectories, nominal models and actual models are drawn randomly from Gaussian Processes. The performance of the batch pseudo-inverse ILC (3.32) is shown in the red line. Numerical stability issues prevent it from stabilizing at steady state error, whereas recursive ILC (blue line) converges stably. If the model mismatch is increased, at some point, recursive ILC also diverges. Applying caution without adaptation is not enough to converge to steady state error. Cautious *and* adaptive *bayesILC*, on the other hand, applying the updates (3.14) and (3.29) iteratively, is very effective and shows a stably convergent behaviour. 51
- 3.5. The proposed ILC algorithm is evaluated on random nonlinear systems. The Frobenius norm of the trajectory deviations, \mathcal{J}_k , is plotted over the iterations k . Results are averaged over ten experiments, where for each experiment, trajectories and dynamics along these trajectories are drawn from Gaussian Processes. Recursive ILC that is not cautious shows an unstable behaviour, and adding adaptation without caution is also not stable (both not shown in the Figure). Purely cautious ILC (red line) is divergent for some of the trajectories. Cautious *and* adaptive *bayesILC*, on the other hand (blue line), shows a stable convergent learning performance. 53
- 3.6. The performance of the adaptive and cautious ILC algorithm *bayesILC* on the simulated Barrett WAM model is shown on the left-hand side. The Frobenius norm of the trajectory deviations, \mathcal{J}_k , is plotted over the iterations k . The results are averaged over ten different strikes and three different initial postures. Three different adaptation laws are considered, adaptation of discrete-time and continuous-time LTV models are shown in blue and red, respectively, while the adaptation of link parameters is shown in black. Forgetting factor was set to $\lambda = 0.8$ for all of the adaptation laws. One of the desired trajectories, shown in dashed red on the right-hand side, is tracked very closely in the final iteration. The blue markers correspond to the time profile of the motion, which are drawn uniformly spaced, one for each 80 milliseconds. The final hitting positions reached are shown as filled circles. 54
- 3.7. Joint trajectories for a hitting movement on the Barrett WAM model. The reference trajectories, shown in dashed red, are tracked very closely with ILC in the final iteration, shown in blue. 55

-
- 3.8. Simulation results illustrating the additional robustness to varying initial conditions whenever the trajectories (states and control references) are adapted according to (3.34) (blue line). Note the unstable performance of ILC without such adaptation (black line), which keeps the references \mathbf{r}_j and the inverse dynamics inputs $\mathbf{u}_{IDM,j}$ fixed. 56
- 3.9. An example of a striking movement for real robot table tennis is shown in red. The blue markers correspond to the time profile of the motion, which are drawn uniformly, one for each 80 milliseconds. Executing this movement well with the Barrett WAM will lead to a good hit. Control errors in tracking lead to a poor hitting performance, shown in blue. The filled circles are the final reached hitting positions. High-gain PD feedback was used to track the reference in this real robot example. The tracking errors can be decreased efficiently and stably by applying the proposed recursive, cautious and adaptive ILC update *bayesILC*. . 57
- 3.10. Two Barrett WAMs (a.k.a. *Ping* and *Pong*) are initialized in our experiments in three different starting postures. We make controlled experiments with a simulated ballgun, and generate many different hitting movements, three of them are shown in the above images. The proposed algorithm *bayesILC* leads to an efficient and stable learning approach for tracking these hitting movements. The right-hand side starting posture for the robot *Ping* can be seen on the upper left image. Initially, before learning with ILC starts, *Ping* performs poorly, and the hitting posture of the robot is shown in the upper central image. After five iterations, the hitting posture is corrected significantly as shown in the upper right image. Similarly, the central images show the operation of the ILC for another trajectory, where the starting posture for *Ping* is fixed on the left-hand side of the robot. On the bottom images, an ILC performance is shown for the robot *Pong*. The three plots on the right-hand side show the Cartesian trajectories corresponding to the ILC iterations. The reference trajectories are shown in dashed red, and the final hitting positions reached are shown as filled circles. 58
- 3.11. Robot experiment results for cautious and adaptive *bayesILC*, shown for a particular reference trajectory. The ten iteration results are concatenated for convenience. The desired joint trajectories correspond to a hitting movement on the Barrett WAM. The reference trajectories, shown in red, are tracked very closely with ILC in the final iteration, shown in blue. Final cost goes down to 0.20 in the last iteration. 59

-
- 4.1. Our robot table tennis setup with a seven DoF Barrett WAM, where we demonstrate, using kinesthetic teach-in, multiple good table tennis serve movements while recording the resulting joint-space robot trajectories. A metal piece is attached to the end effector of the Barrett WAM, which connects to a standard sized table tennis racket. An egg-holder on the metal piece holds the ball initially before the serve. The demonstrator, after finding a good starting posture, starts by swinging the arm, giving the ball enough acceleration to propel it away from the robot. The ball is then hit in midair by a careful adjustment of the robot wrist. The initial posture, the swinging movement of the robot shoulder joints and the elbow, and finally the turning of the wrist all contribute to the style of the shown movement. Multiple demonstrations starting from different initial postures are recorded in one session. We compare and evaluate throughout the chapter different learning from demonstrations approaches using these demonstrations. We propose a new iterative optimization approach that can learn sparse parameters while adapting the features of the movement primitives to the demonstration data. 63
 - 4.2. An example Elastic Net path with twenty selected parameters is shown after training Algorithm 6, *cLSDP*, with five demonstrations. This *regularization path* can be generated in the final step of the algorithm. As the l_1 -penalty term λ_1 of the regression problem (4.1) is reduced, the coefficients converge to their (maximal) ordinary least squares values at the right hand side of the plot (not shown). Each dashed line signals a change in the regularization term, and the coefficients are updated accordingly. The algorithm *LARS* [4] can be used to generate these piece-wise linear regularization paths. One possible way to use this path is to rank the sparse parameters of the learned movement primitives in terms of statistical importance. For example, in the shown plot, the parameters corresponding to the red lines would be ranked after the other parameters appearing before (black lines). The parameter paths, whose coefficients become nonzero close to each other, are drawn with the same color. 69
 - 4.3. Two movement primitives learned by the proposed algorithm *cLSDP*, are plotted in joint space against the recorded demonstrations. The table tennis serve movements, shown in blue, after preprocessing and segmenting the recorded time series are one second long each. The first three rows, q_1 through q_3 , correspond to the shoulder movement in joint space. The fourth row q_4 shows the movement of the elbow. Finally, the last three rows (q_5 through q_7) show the wrist movements in joint space. The trained movement primitives, shown in orange, couple the sparse regression parameters across the degrees of freedom of the robot. . . . 70
 - 4.4. Three example demonstrations in task space. The initial position of the racket center and the ball in the egg-holder are marked as 0 in red and blue circles, respectively. The egg-holder is located approximately 14 cm away from the racket centre. Before the racket stops moving, the ball is already hit, flying towards the table. 72
 - 4.5. A successful rollout during real robot experiments. The ball is initially on top of the egg-holder and during the movement, as a result of the acceleration of the arm, it takes-off from the robot, to be later hit by the racket towards the table. The arm then decelerates towards a safe resting posture. 72
-

A.1. Using Extended Kalman Smoothing (EKS) to estimate the parameters of the rebound model from actual noisy ball data during a demonstration recording. Ball observations are acquired from two different sets of cameras on opposite sides of the table, shown as red and blue circles respectively. They are then smoothed with the EKS, shown in yellow, to obtain velocity estimates before rebound and just after rebound. Nonlinear least squares is then used to estimate the rebound model parameters μ_t and ϵ_t	86
A.2. Kalman Filtering with simultaneous outlier detection. The ball detection algorithm sometimes outputs outliers, typically more as the ball approaches the racket. Such corrupted data can be identified and discarded using the covariance of the Kalman Filter.	87

List of Tables

2.1. Table of Symbols	11
2.2. Results comparing FP and VHP	25
4.1. Comparison of different learning from demonstrations approaches, averaged over five different serve demonstrations	71
A.1. Ball model parameter estimates	86

Abbreviations, Symbols and Operators

1 Introduction

Although available computational power has increased tremendously over the last years, the current state of the art in robotics is still far away from allowing the deployment of autonomous robots in complex human-inhabited environments. Faced with many uncertainties and complex set of interactions in such environments, the robotics research is heavily focusing on extending the classical robotics and control paradigms. The algorithms developed in classical robotics and control are suitable only for nonchanging, predictable industrial environments. While the recent growing interest in machine learning allows for prediction and control in changing and noisy environments, it is still not clear how to integrate black box prediction and learning algorithms in robotics, where safety is of paramount importance. The recent advances in model-free Reinforcement Learning in Go and Atari playing, for example, do not transfer (unfortunately) in a straightforward manner to advances in robotics.

For complex tasks and interactions with the environment, it is an assumption of this thesis that robots need to learn good models, and they need to learn to use them wisely. As will be evident from the content of this thesis, optimization becomes an important tool both in the search for good models from data, and in the use of them for planning and control. The unifying thread throughout the thesis is that optimization plays an important role in all of the different components of robotics, from planning, trajectory generation and control to learning. Implemented as a set of efficient numerical routines, optimization allows us to implement learning, control and planning algorithms. The interactions between them, when applied to the dynamic task of table tennis, is what constitutes the contents of this thesis.

1.1 Motivation

Our motivation for the research conducted in this thesis comes from table tennis, a challenging game that is easy to learn for humans but difficult to master. Robot table tennis is also a challenging task for the current robots, limited in terms of both hardware and software. Table tennis shows complexities that interact with each other in nontrivial ways: from the camera vision to estimating a ball reliably, and from filtering, prediction of a spinning ball to precise robot control, the dynamic task offers many opportunities for advancing the state of the art. We think that the successful imitation and assimilation of human dexterous motor skills and understanding the inherent perception-action loop better will pave the way for future advances in robotics.

Robot table tennis has, since the eighties, captivated the attention of the robot control and learning communities as a challenging and dynamic task, and research in it has been ongoing ever since. After the pioneering work of Anderson's analytical player [5] and the early engineering successes during the robot table-tennis competitions [6], [7], there have been various research focusing on different aspects of the game. These include, but are not limited to, simplifications in trajectory generation using a virtual hitting plane [8], [9], improvements in the overall robotic setup [10], [11], or learning approaches to generate better strikes with Reinforcement Learning (RL) [12], [13]. For a more complete summary, please see Section 2.2.

Learning and control algorithms that efficiently improve the performance in a robot table tennis setup are considered throughout the thesis. The thesis first focuses exclusively on the trajectory generation part in Chapter 2. The optimization framework developed is then used in Chapter 3, where these trajectories are tracked with a new learning control approach. The optimization approaches considered in Chapter 2 were all tested against a ballgun, and as a remedy, we explore in Chapter 4 how a robot could learn to serve a ball autonomously by learning from demonstrations. In this setting, we record multiple human demonstrations during a kinesthetic teach-in recording and apply a new approach to learn sparse parameters while adapting the basis functions to the data. Moreover, these parameters are ranked in terms of their task importance, which is desirable for future reinforcement learning applications.

1.2 Contributions

In this section, we summarize the contributions of the thesis. The algorithms introduced in Chapter 2, and evaluated in our real robot table tennis setup, extend the state of the art in robot table tennis. The learning control algorithm introduced in Chapter 3, using the passive learning framework from dual control [14], contributes to the Iterative Learning Control literature. The alternating optimization considered in Chapter 4 is a novel application of multi-task Elastic-Net [15] regression to feature selection in across multiple degrees of freedom robot joint recordings.

1.2.1 Algorithms

We introduce two new algorithms/table tennis players in Chapter 2. The first table tennis player is called the *Focused Player*, or FP in short. As a result of the additional equality constraints in the optimization, the Focused Player tries to return the balls to a desired position (at a desired time) to the opponent’s court. The second player, called the *Defensive Player* (DP), leads to a more defensive play style. With the introduction of the additional resting posture optimization, the Defensive Player leads to larger accuracy in ball returning performance.

These optimized trajectories need to be tracked accurately in order to increase returning performance further. For this purpose, we introduce a new Iterative Learning Control (ILC) algorithm, called *bayesILC*, that uses a Bayesian model-based approach. The Bayesian framework allows us to model uncertainty and to exercise caution during learning. Chapter 3 discusses how we can implement cautious and adaptive behavior at the same time in recursive ILC updates. Linear Bayes Regression is used to adapt the nominal model matrices, initially acquired via linearizing a nominal forward dynamics model. We show and discuss throughout Chapter 3 how by selecting a suitable forgetting factor, we can learn the iteration dependent robot models (more precisely, derivatives of the forward dynamics model that describe the behavior of the robot only around the reference) more flexibly, increasing the tracking performance.

In Chapter 4, learning a sparse representation of robot demonstrations is considered in a novel optimization framework. First, learning from a single demonstration is explored, where the features are enforced across the multiple degrees of freedom (DoF) of the robot. An iterative optimization algorithm, called *LSDP*, is suggested, where multi-task Elastic Net regression is alternated with a nonlinear optimization (BFGS, in particular, is applied) on the radial basis function parameters. The weighted multi-task Elastic Net considered penalizes in particular

the accelerations of the resulting movement primitive as well as the l_1 -norm of the regression coefficients. The resulting sparsity can be exploited to adapt the basis functions more effectively. A variant of the approach, called coupled *LSDP* or *cLSDP* for short, shares the features not across the multiple DoFs of the robot, but across multiple demonstrations. Algorithm *cLSDP* requires more parameters for the basis functions (independently adapted for each DoF) but couples the regression parameters across the degrees of freedom. Most importantly, the resulting number of parameters in the learning procedure is then independent of the degrees of freedom of the robot.

1.2.2 Applications

Algorithms introduced throughout the thesis are evaluated both in simulation and in our real robot table tennis platform, see Figure 4.1. In our robot table tennis experiments in Chapter 2, we use a seven DOF Barrett WAM capable of fast accelerations and velocities. The robot is torque-controlled and cable driven. In order to detect incoming balls, we used four cameras hanging from each corner of the ceiling. After the detection and the triangulation of the incoming ball positions, we use an Extended Kalman Filter with a spinning ball model to filter the ball state. The Kalman Filtering framework requires initial means and variances of the state. One of the contributions of Chapter 2 is the estimation of the initial ball positions and velocities using online optimization. A topspin component parameterizing the spinning ball model is estimated in addition. The variance of the ball state is used throughout the experiments in an upper-confidence bound based criterion to reject outliers. The outliers, given sometimes by our vision system, can otherwise completely ruin the accuracy of the Kalman Filter.

Throughout the experiments in Chapter 2, we use high gain PD control to track the computed reference trajectories. Tested against a ballgun that is moved around the table, we find that the high gain PD control can track slower and shorter trajectories more accurately. Whenever the robot has to move more or faster inside the workspace to hit the ball, the percentage of the balls returned successfully to the other side drops sharply. This motivates the development of the model-based Iterative Learning Controller in Chapter 3. The Bayesian ILC algorithm *bayesILC* is implemented online to learn to track the hitting movements accurately. We show in real robot experiments, the improvement of the tracking performance over the iterations, averaged over many different hitting trajectories. Linear time varying models of the Barret WAM are adapted over the iterations, improving the performance further.

In Chapter 4, the developed learning from demonstrations framework, and in particular the Algorithm *cLSDP* is tested in a ball serving task. In our preliminary real robot experiments, we show a successful serve that uses our new movement primitive representation. The sparse regression parameters can be ranked using the Elastic-Net path of coefficients (as a function of regularization), which we note is a desirable feature to combat the curse of dimensionality in Reinforcement Learning applications.

1.3 Outline

The three chapters that follow this introduction can be read independently, as they consider different subtasks of table tennis. The problems considered in each chapter could be seen perhaps, as slices through the underlying challenging robotics task. In Chapter 2, we consider trajectory

generation for robot table tennis using a free final-time optimal control approach. We introduce two new robot players that are based on two different optimizers run online. The algorithms lead to two different play styles, and either can be considered within a higher-level strategy.

In Chapter 3, we consider tracking these kinematics-level (i.e., desired joint positions, velocities and accelerations) trajectories accurately by learning the appropriate torque commands. We extend a recursive and model-based approach to Iterative Learning Control (ILC) by using a Bayesian framework. The means and variances of our parameterized models are learned with Linear Bayesian Regression. Variances of the models are directly used in the recursive ILC update, as effective regularization terms of the control input updates. We can implement cautious learning within this framework by initializing the variances with large values. Our cautious and adaptive ILC algorithm is tested in the Barrett WAM and discussions are given on different forms of model adaptation.

In Chapter 4, we focus on learning from demonstrations and develop a new framework for extracting sparse representations of demonstrated robot movements. The developed novel approach, called *LSDP*, and its coupled variant *cLSDP* for multiple-demonstrations, can be used to learn sparse representations while having low accelerations throughout the movement. A sparse representation can potentially be exploited in a Reinforcement Learning setting. The sparse parameters can be ranked by following their Elastic Net path, and for *cLSDP* they are independent of the robot degrees of freedom.

In the final chapter, we conclude by mentioning the open problems and future work that can extend the contributions of the thesis. The appendices include the technical details not mentioned in the previous chapters.

2 Online Optimal Trajectory Generation for Robot Table Tennis

Table tennis is a challenging game for humans to master. For robots, it also serves as a testbed to study and validate the effectiveness of different movement generation algorithms. Combining different estimation, movement generation and execution schemes and studying how close they come to imitating expert human behaviour will yield important insights for robotics research.

2.1 Introduction

Optimality plays an important role in the search for efficient and feasible striking trajectories. However, so far most of the research in robotic table tennis were based on specialized systems, such as Cartesian coordinate robots [16, 11], that eliminate great part of the difficulties in trajectory generation. Furthermore, most algorithms for robotic table tennis focused on simplifications of the game that reduced the dimensions of the search space [9] in order to quickly come up with a movement plan. In this chapter, we show the advantages of incorporating optimality in trajectory generation to create more flexible movement.

Our robotic setup with an anthropomorphic seven degree of freedom Barrett WAM arm is shown in Figure 4.1. The redundant arm can achieve high speeds and accelerations. It is a good platform to study different movement generation schemes. Optimal control based approaches have the potential to make use of all degrees of freedom in planning, contributing to more natural and efficient generation of strikes. In this chapter we introduce an optimal control framework in robot table tennis where the generation of striking trajectories is the result of an optimization problem. As opposed to previous works, inverse kinematics or a fixed plane to compute joint trajectories are not needed. Two different optimization approaches are presented that encode defensive and goal-oriented styles of playing. We show extensive experiments in simulation and on our table tennis platform, where we evaluate and compare the performance of the algorithms. We do not rely on pure physical modeling to compute desired ball and racket parameters. Instead, the parameters of the prediction models are estimated based on offline human ball-racket demonstrations and the angular velocity (spin) of the ball is estimated online from actual ball data.

In the remainder of this chapter, the framework is described in detail. A brief survey of robot table tennis research is given and related work on trajectory generation is introduced in Section 3.1.1. Robot trajectory generation for table tennis is formalized as an optimal control problem in Section 2.3. Two efficient solvers are presented in Sections 2.4 and 2.5 for optimizing the cost functional under additional constraints. The performance of the two resulting players are evaluated in Section 4.4 and it is shown that they compare favorably with an inverse kinematics based approach in simulation. Finally, real robot experiments are performed, where the algorithms run online in the table tennis setup. Based on this evaluation, conclusions are given with several promising extensions which might be necessary to increase performance further.



Figure 2.1: Robotic table tennis setup with four cameras on the corners of the ceiling tracking the ball at 60 Hz. We present two optimal control based trajectory generation algorithms that encode defensive and goal-oriented styles of playing. A constrained nonlinear optimization problem is solved in both cases to find an optimal striking trajectory as well as an optimal striking time.

2.2 Related Work

Robot table tennis started as a challenge [6] and Anderson was the first in 1988 to construct a table tennis playing robot [5]. Dexterous motion displayed by expert table tennis players as well as the challenges in accurate ball state prediction piqued the curiosity of robot researchers. Since 1988, interest in robot table tennis has continued with various robotic platforms, for example, [17] and [10]. Earlier Cartesian coordinate robots ([16, 11], among others) were followed by industrial arms and humanoid robots with a seven degrees of freedom arm (e.g., [18, 19, 20]). Different control techniques for humanoid table tennis robots were proposed in [21] and [20]. A comprehensive categorization and summary of robot table tennis research was given in [22].

Research in robot table tennis considered ball estimation and prediction algorithms as well. Physical flight models without spin were considered in [16, 9]. Flight, rebound and racket contact models incorporating spin effects were proposed, for example, in [23] and in [24]. Frameworks estimating spin from cameras include [25] and [26]. Recently, a framework for estimating the spin of the table tennis ball using offline clustering and an online Expectation Maximization based state estimation algorithm was introduced in [27]. The authors argue that the change of spin is very slow and they assume spin to be constant.

One of the most popular frameworks for trajectory generation in table tennis is the Virtual Hitting Plane (VHP) method, which is based on the virtual hitting point hypothesis [8]. In this approach, the trajectory of an incoming ball is first estimated from a stream of ball position observations. Usually, a physical flight model is then used to predict the intersection point of the future ball trajectory with an appropriately chosen plane. This procedure determines the striking time as well as the striking point. The remaining task-space parameters, the desired racket velocity and normal at striking time, are determined by running the physical flight model backwards from a desired ball landing position and velocity, and inverting the ball-racket contact model. For a more general discussion, see [16] and [9]. A clear limitation of the method is shown in Figure 2.2. A player fixing the VHP may not generate feasible trajectories for some ball trajectories. By means of trajectory optimization, trajectories can be generated that are not constrained to a hitting plane.

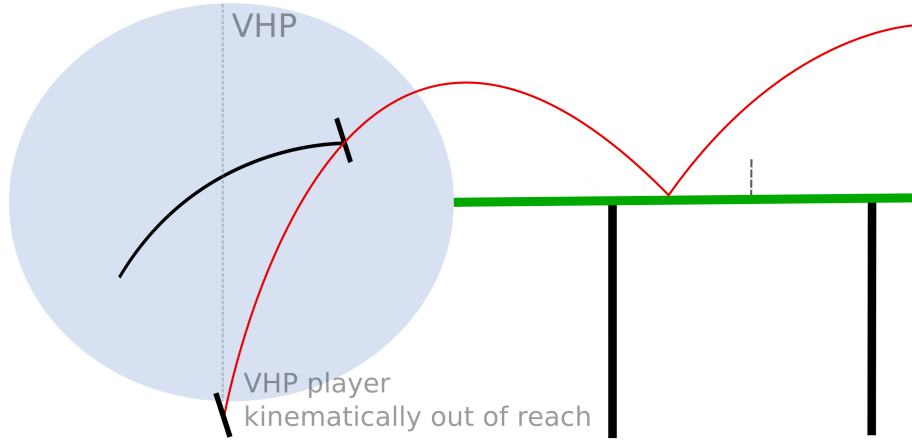


Figure 2.2: Fixing a virtual hitting plane (VHP) can make the generated trajectories unnecessarily restrictive and the resulting inverse kinematics may be infeasible. Instead the whole ball trajectory should be considered in a trajectory generation framework and the hitting time as well as the hitting point should be optimized. The predicted ball trajectory and a feasible racket trajectory are shown in red and black, respectively. VHP is shown as a dotted gray line, the workspace of the robot is shown as an ellipsoidal light blue region.

Another framework that uses a mixture of movement primitives and reinforcement learning (RL) is given in [12]. Initialized with a set of dynamic movement primitives (DMP) extracted from demonstrations, RL is applied to select and generalize between the teach-in movements. A problem with this approach is that not all robots can be trained well this way. For example, the shoulder of the robot shown in Figure 4.1 weighs 10 kg alone, and the wrist weighs about 2.5 kg. It is more difficult to move the links with heavy inertia, whereas it is easier to find optimization algorithms that make use of them. A different approach in [13] uses RL to learn robot movements as a response to predicted ball trajectories. The learned ball trajectories are second order polynomials, restricting the validity of the proposed approach to the front side of the robot workspace.

A related dynamic framework involving moving targets is the ball catching robot of [28] where a computationally demanding optimization problem is solved online. It includes also the catching time as another parameter to be optimized. The framework of [29] considers generating catching movements for more general objects. Another application of optimal control showing the benefits of spatio-temporal optimization is given in [30] on a brachiating robot. The computed solutions require lower torques when compared with traditional optimal control approaches fixing the time interval.

2.3 Problem Statement

Most of the algorithms for robotic table tennis need to specify when, where and how to intercept the incoming ball trajectory $\mathbf{b}(t)$. In [16] and [9] for example, the authors calculate the intersection point of a predicted ball trajectory $\mathbf{b}(t)$ with a virtual hitting plane (VHP) at $y = y_{\text{VHP}}$ to determine the space and time coordinates of the hitting event. Although additional constraints like the VHP can simplify trajectory generation, they can also lead to awkward or

infeasible movements. It is possible to eliminate this plane altogether and include the striking time as another parameter to be determined in an optimization problem.

When generating striking trajectories for a robot with n degrees of freedom, trajectories with minimal acceleration can be preferred for safety and efficiency reasons. Consider the following *free-time* optimal control problem [31]

$$\min_{\mathbf{\ddot{q}}, T} \int_0^T \mathbf{\ddot{q}}(t)^T \mathbf{\ddot{q}}(t) dt \quad (2.1)$$

$$\text{s.t. } \Psi_{\text{hit}}(\mathbf{q}(T), T) \in \mathcal{H}, \quad (2.2)$$

$$\Psi_{\text{net}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) \in \mathcal{N}, \quad (2.3)$$

$$\Psi_{\text{land}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) \in \mathcal{L}, \quad (2.4)$$

$$\mathbf{q}(0) = \mathbf{q}_0, \quad (2.5)$$

$$\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}_0, \quad (2.6)$$

where the final hitting time T is an additional variable to be optimized along with the joint accelerations $\mathbf{\ddot{q}}(t): [0, T] \rightarrow \mathbb{R}^n$. Initial conditions for the robot are the joint positions \mathbf{q}_0 and joint velocities $\dot{\mathbf{q}}_0$. The inequality constraints (2.2) – (2.4) ensure that the task requirements for table tennis are satisfied. The hitting constraint $\Psi_{\text{hit}} \in \mathcal{H}$ ensures impact of the racket with the ball at striking time T . The net constraint $\Psi_{\text{net}} \in \mathcal{N}$ makes sure the ball passes over the net and finally, the landing constraint $\Psi_{\text{land}} \in \mathcal{L}$ captures the requirement that the ball should bounce first on the opponents court. See Figure 2.3 for an illustration. The precise definitions of these constraint functions and the constraint sets will be introduced in section 2.5.

Solutions of (2.1) – (2.6) can be found using Pontryagin's minimum principle [32]. The optimal $\mathbf{q}(t)$ is a third degree polynomial for each degree of freedom, with the inequality constraints (2.2) – (2.4) imposing generalized *transversality conditions* on the Hamiltonian and the momenta to satisfy at striking time [33, 34]. Solving such boundary value problems is hard, especially given real time constraints. In the later sections we will introduce two algorithms that will solve this problem efficiently under additional constraints. These two approaches can be seen as different ways to solve the underlying table tennis task efficiently and they lead to two different play-styles.

When given only constraints at the boundary, the striking time T , the joint position and velocity values at striking time \mathbf{q}_f and $\dot{\mathbf{q}}_f$ fully parametrize this problem. The polynomial coefficients for the striking trajectory

$$\mathbf{q}_{\text{strike}}(t) = \mathbf{a}_3 t^3 + \mathbf{a}_2 t^2 + \dot{\mathbf{q}}_0 t + \mathbf{q}_0, \quad (2.7)$$

can then be determined in joint-space for each degree of freedom of the robot

$$\begin{aligned} \mathbf{a}_3 &= \frac{2}{T^3}(\mathbf{q}_0 - \mathbf{q}_f) + \frac{1}{T^2}(\dot{\mathbf{q}}_0 + \dot{\mathbf{q}}_f), \\ \mathbf{a}_2 &= \frac{3}{T^2}(\mathbf{q}_f - \mathbf{q}_0) - \frac{1}{T}(\dot{\mathbf{q}}_f + 2\dot{\mathbf{q}}_0). \end{aligned} \quad (2.8)$$

The notation that is used frequently in the rest of the chapter is shown in Table 2.1 for the reader's convenience.

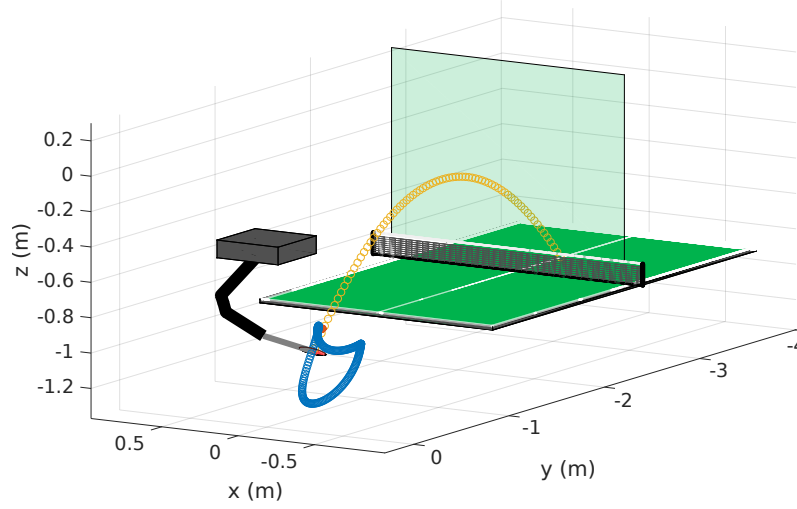


Figure 2.3: In table tennis, robot trajectories (blue) can be seen as reactions to predicted ball trajectories. The players are free to decide where, when and how to intercept the ball. However, the resulting outgoing ball trajectories (orange) need to be *feasible*: the ball has to pass above the net and land on the opponent's court. The feasible region above the net is drawn in transparent green. The rules of the game can be captured as constraints for generating robot striking trajectories.

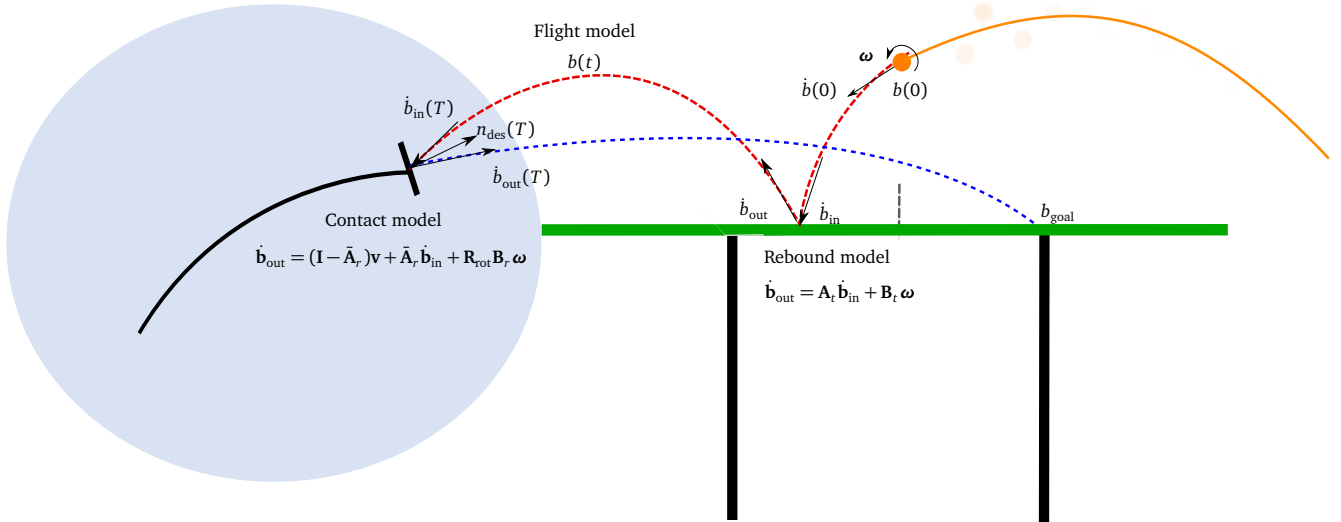


Figure 2.4: Ball prediction schema for table tennis. After estimating the initial ball position, velocity and spin, the future path of the ball can be predicted using the flight model, the rebound model and the racket-ball contact model. The trajectory generation framework uses these models to compute desired striking trajectories.

2.3.1 Background on Ball Prediction

For the trajectory generation process, three ball models will be used to determine the table tennis task constraints (2.2) – (2.4): ball flight model, ball-table rebound model and ball-racket contact model. Whenever an incoming ball is detected in midair, a flight model will first be used to predict the trajectory $\mathbf{b}(t)$ of the ball centre of mass coordinates $\mathbf{b} = (b_x, b_y, b_z)^T$ until impact with a racket, table or ground.

Table 2.1: Table of Symbols

Notation	Explanation
T	hitting time
T_{rest}	return time
T_{land}	desired ball landing time after hit
\mathbf{b}_{goal}	desired ball landing positions
$\mathbf{q}(t)$	joint trajectory
$\mathbf{b}(t)$	predicted ball trajectory
$\mathbf{r}(t)$	racket center position
$\mathbf{v}(t)$	racket velocity
$\mathbf{n}(t)$	racket normal
\mathbf{K}_p	kinematics function for racket center position
\mathbf{K}_n	kinematics function for racket normal
$\mathbf{J}(\mathbf{q}_f)$	jacobian at hitting time
$\mathbf{n}_{\text{des}}(T)$	desired racket normal at hitting time
$\mathbf{v}_{\text{des}}(T)$	desired racket velocity at hitting time
$\boldsymbol{\omega}$	ball spin
$\dot{\mathbf{b}}_{\text{in}}, \dot{\mathbf{b}}_{\text{out}}$	ball velocity before and after impact
N	minimum number of balls to start prediction
$\mathbf{q}_0, \dot{\mathbf{q}}_0$	initial joint positions and velocities
$\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$	joint position and velocity estimates
$\mathbf{q}_f, \dot{\mathbf{q}}_f$	joint position and velocity at hitting time
\mathbf{q}_{ext}	joint extreme values of trajectory
$\mathbf{q}_{\text{max}}, \mathbf{q}_{\text{min}}$	joint angle upper and lower limits
\mathbf{R}	weighting matrix
$\mathbf{q}_{\text{strike}}(t)$	joint striking trajectory
$\mathbf{q}_{\text{return}}(t)$	joint returning trajectory
$\Psi_{\text{hit}}, \Psi_{\text{land}}, \Psi_{\text{net}}$	table tennis task constraints

Flight model

Table tennis balls are very light, a standard ball weighs about 2.7 grams, which makes nonlinear effects due to air drag and spin noticeable especially when the ball speed $v = \|\dot{\mathbf{b}}\|_2$ is high. The *flight model* [23]

$$\ddot{\mathbf{b}} = \mathbf{g} - C_D v \dot{\mathbf{b}} + C_L \boldsymbol{\omega} \times \dot{\mathbf{b}}, \quad (2.9)$$

is a nonlinear dynamics model that incorporates air drag and spin effects. The air drag constant C_D and the lift constant C_L as well as gravity g , $\mathbf{g} = (0, 0, g)^T$, parameterize this model. The *magnus effect* due to spin (angular velocity) $\boldsymbol{\omega}$, for example, acts as an additional downward force for an incoming ball if the angular velocities are in the negative x -direction (topspin). It is assumed that spin stays constant throughout the ball motion.

Rebound Model

Formally, rebound is a discrete event which reflects the ball velocity when the ball hits the table. The incoming velocities $\dot{\mathbf{b}}_{\text{in}}$ at bouncing time are transformed to outgoing velocities $\dot{\mathbf{b}}_{\text{out}}$. The following nonlinear *rebound model* [23] for a standard ball with radius $r_B = 2$ cm,

$$\dot{\mathbf{b}}_{\text{out}} = \mathbf{A}_t \dot{\mathbf{b}}_{\text{in}} + \mathbf{B}_t \boldsymbol{\omega}, \quad (2.10)$$

is parameterized by the dynamic coefficient of friction μ_t and the coefficient of restitution ϵ_t of the table

$$\mathbf{A}_t = \begin{bmatrix} 1-\alpha & 0 & 0 \\ 0 & 1-\alpha & 0 \\ 0 & 0 & -\epsilon_t \end{bmatrix}, \quad \mathbf{B}_t = \begin{bmatrix} 0 & \alpha r_B & 0 \\ -\alpha r_B & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad (2.11)$$

where the nonlinearity comes from the sliding parameter

$$\alpha = \mu_t (1 + \epsilon_t) \frac{\dot{b}_z}{\|\dot{\mathbf{b}}_T\|}, \quad (2.12)$$

$\dot{\mathbf{b}}_T$ is the *tangent velocity* at contact

$$\dot{\mathbf{b}}_T = (\dot{b}_x - r_B \omega_y, \dot{b}_y + r_B \omega_x, 0)^T, \quad (2.13)$$

for $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^T$. This model suggests, for example, that some amount of topspin is transferred at sliding impact to linear velocity in the y -direction. See Figure 2.4 for a table tennis schema.

Racket Contact Model

We assume the following linear racket contact model holds for a standard racket with radius $r_R \approx 7.6$ cm,

$$\mathbf{o} = \mathbf{A}_r \mathbf{i} + \mathbf{B}_r \boldsymbol{\omega}, \quad (2.14)$$

between the outgoing ball velocity \mathbf{o} and the incoming ball velocity \mathbf{i} , similar to (2.10) but in the moving racket frame. The outgoing ball Cartesian velocities are hence found by multiplying \mathbf{o} with the racket rotation matrix and adding the racket velocities, i.e., $\dot{\mathbf{b}}_{\text{out}}(t) = \mathbf{R}_{\text{rot}} \mathbf{o}(t) + \mathbf{v}(t)$, where the rotation matrix $\mathbf{R}_{\text{rot}}(\mathbf{q}(t))$ of the racket is given by the kinematics function. The impact model is parameterized by the constants κ and ϵ_r

$$\mathbf{A}_r = \begin{bmatrix} 1 - \kappa & 0 & 0 \\ 0 & 1 - \kappa & 0 \\ 0 & 0 & -\epsilon_r \end{bmatrix}, \quad \mathbf{B}_r = \begin{bmatrix} 0 & \kappa r_R & 0 \\ -\kappa r_R & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \quad (2.15)$$

Letting $\bar{\mathbf{A}}_r := \mathbf{R}_{\text{rot}} \mathbf{A}_r \mathbf{R}_{\text{rot}}^T$, we get the following relationship between the Cartesian velocities:

$$\dot{\mathbf{b}}_{\text{out}} = (\mathbf{I} - \bar{\mathbf{A}}_r) \mathbf{v} + \bar{\mathbf{A}}_r \dot{\mathbf{b}}_{\text{in}} + \mathbf{R}_{\text{rot}} \mathbf{B}_r \boldsymbol{\omega}. \quad (2.16)$$

Ball Prediction

The models (2.9) – (2.16) can be composed together to predict the future ball trajectory given camera observations. We use an Extended Kalman Filter (EKF) to estimate the ball state from observations [35]. The ball state for the filter is the ball positions and velocities, since we assume that the ball spin is constant throughout motion. The ball spin can be seen as a parameter of the prediction functions.

EKF instantiated with the models (2.9) – (2.16), the initial ball positions \mathbf{b}_0 and velocities $\dot{\mathbf{b}}_0$ and spin $\boldsymbol{\omega}$, estimates the evolving ball state and predicts the future ball trajectory at each time instant t : a multivariate normal distribution $p_t(\mathbf{b}, \dot{\mathbf{b}})$ of ball states parameterized by time is generated

$$(\mathbf{b}(t)^\top, \dot{\mathbf{b}}(t)^\top)^\top \sim p_t(\mathbf{b}, \dot{\mathbf{b}}) = \mathcal{N}(\boldsymbol{\mu}(t), \boldsymbol{\Sigma}(t)), \quad (2.17)$$

where $\boldsymbol{\mu}(t) = (\mathbf{b}(t)^\top, \dot{\mathbf{b}}(t)^\top)^\top$ is the mean ball position and velocity predictions. The covariance matrix $\boldsymbol{\Sigma}(t)$ is updated along with the mean estimate $\boldsymbol{\mu}(t)$ using the EKF predict and update equations. The covariance matrices are used to reject outliers and hence make Kalman Filtering more robust to ball detection errors.

2.4 The Focused Player

A higher-level strategy in table tennis could, based on a perceived state of the opponent, command to return an incoming ball to a desired location. A reliable trajectory generation algorithm for that purpose should be flexible and easily find safe joint movements. The optimal control based approach penalizing sum of squared accelerations, in this regard, leads to a flexible optimization problem where it is easy to find good hitting postures, while satisfying additional safety constraints.

2.4.1 Racket Constraints

For a table tennis player that wants to guarantee the return of the incoming ball to a desired location at a desired landing time, the optimal control problem introduced in (2.1) can be solved efficiently under additional racket constraints

$$\mathbf{K}_p(\mathbf{q}(T)) = \mathbf{b}(T), \quad (2.18)$$

$$\mathbf{K}_n(\mathbf{q}(T)) = \mathbf{n}_{\text{des}}(T), \quad (2.19)$$

$$\mathbf{J}(\mathbf{q}(T))\dot{\mathbf{q}}(T) = \mathbf{v}_{\text{des}}(T). \quad (2.20)$$

The racket center position $\mathbf{r}(T)$ and the racket normal $\mathbf{n}(T)$ at hitting time T are computed using the kinematics functions $\mathbf{K}_p(\cdot)$ and $\mathbf{K}_n(\cdot)$, respectively. The Jacobian $\mathbf{J}(\cdot) \in \mathbb{R}^{3 \times n}$ [36] at hitting time transforms the joint velocities in (2.20) to racket velocities $\mathbf{v}(T)$. To maximize the probability of hitting the ball, the desired racket center is set at hitting time equal to the mean ball position estimate in (2.18), i.e., $\mathbf{r}(T) = \mathbf{b}(T)$. To return the ball to the opponent's court, the constraints on the racket normal $\mathbf{n}(T)$ and velocity at hitting time $\mathbf{v}(T)$ are imposed in (2.19) and (2.20), respectively.

The imposed racket constraints (2.18) – (2.20) can satisfy and hence effectively replace the table tennis task constraints (2.2) – (2.4) for suitable $\mathbf{n}_{\text{des}}(T), \mathbf{v}_{\text{des}}(T)$, if the future path of the incoming ball is predicted before the optimization for calculating the hitting movement takes place.

Calculating Desired Racket Parameters

After predicting the future ball path $\mathbf{b}(t)$ at a discrete set of time instants $t \in (0, T_{\text{pred}})$, the next step is to compute desired racket velocities $\mathbf{v}_{\text{des}}(t)$ and desired racket normals on this path $\mathbf{n}_{\text{des}}(t)$. These desired racket parameters will give the incoming ball during the impact, a desired outgoing ball velocity according to (2.16). They are calculated by first specifying a desired landing point \mathbf{b}_{goal} and a desired duration of flight after strike T_{land} . A desired ball outgoing velocity is then found by solving the boundary value problem for the flight model (2.9) with the boundary values

$$\begin{aligned} \mathbf{b}_{\text{out}}(0) &= \mathbf{b}(t), \\ \mathbf{b}_{\text{out}}(T_{\text{land}}) &= \mathbf{b}_{\text{goal}}, \end{aligned} \quad (2.21)$$

for each t . Spin $\boldsymbol{\omega}$ is assumed to be constant throughout. Afterwards, $\mathbf{v}_{\text{des}}(t)$ and $\mathbf{n}_{\text{des}}(t)$ are calculated by inverting the racket contact model (2.16) given the outgoing ball velocities $\dot{\mathbf{b}}_{\text{out}}$ at impact.

In practice, (2.21) can be solved very fast for each t with a gradient-based optimizer. The desired outgoing ball velocities for the sequence of boundary value problems in (2.21) can be initialized with the previous solutions. The closed form solution of the ballistic flight model (i.e., zero drag and spin) can be used to initialize the process.

2.4.2 Nonlinear Constrained Optimization

We briefly show here that the solution $\mathbf{q}(t)$ to the optimal control problem posed in (2.1) under additional racket constraints (2.18) – (2.20) is a third order polynomial for each degree of freedom, $i = 1, \dots, n$.

Derivation from Minimum Principle

Using the minimum principle for unconstrained inputs $\mathbf{u}(t) = \ddot{\mathbf{q}}(t) \in \mathbb{R}^n$, the Hamiltonian

$$\mathcal{H}(\mathbf{u}, \dot{\mathbf{q}}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{u}^T \mathbf{u}(t) + \boldsymbol{\lambda}^T \dot{\mathbf{q}} + \boldsymbol{\mu}^T \mathbf{u} \quad (2.22)$$

for the momenta $[\boldsymbol{\lambda}(t), \boldsymbol{\mu}(t)] \in \mathbb{R}^{2n}$ is minimized at

$$\mathbf{u}^*(t) = -\frac{1}{2}\boldsymbol{\mu}^*(t). \quad (2.23)$$

Costate equation for the momenta gives

$$\begin{aligned} \dot{\boldsymbol{\lambda}}^*(t) &= \mathbf{0}, \\ \dot{\boldsymbol{\mu}}^*(t) &= -\boldsymbol{\lambda}^*(t), \end{aligned} \quad (2.24)$$

or in other terms, $\boldsymbol{\lambda}^* = 12\mathbf{a}_3$, $\boldsymbol{\mu}^* = -2(6\mathbf{a}_3 t + 2\mathbf{a}_2)$, for some constant vectors $\mathbf{a}_3, \mathbf{a}_2 \in \mathbb{R}^n$. Plugging it into (2.23) we get

$$\ddot{\mathbf{q}}^*(t) = 6\mathbf{a}_3 t + 2\mathbf{a}_2, \quad (2.25)$$

which shows that the optimal accelerations are linear functions of time. The joint positions $\mathbf{q}(t)$ are then third order polynomials as in (2.7) with $2n$ coefficients to be determined using $\mathbf{n}_{\text{des}}(T), \mathbf{b}(T), \mathbf{v}_{\text{des}}(T)$ and free final time T as another variable. The transversality condition resulting from the boundary constraints $\Psi(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \mathbf{0}$ can be written as

$$\begin{bmatrix} -\mathcal{H}(T) \\ \boldsymbol{\lambda}(T) \\ \boldsymbol{\mu}(T) \end{bmatrix} = D\Psi^T \boldsymbol{\nu} = \begin{bmatrix} D_T \Psi & D_{\mathbf{q}} \Psi & D_{\dot{\mathbf{q}}} \Psi \end{bmatrix}^T \boldsymbol{\nu}, \quad (2.26)$$

$$\Psi = \begin{bmatrix} \mathbf{K}_p(\mathbf{q}(T)) - \mathbf{b}(T) \\ \mathbf{K}_n(\mathbf{q}(T)) - \mathbf{n}_{\text{des}}(T) \\ \mathbf{J}(\mathbf{q}(T))\dot{\mathbf{q}}(T) - \mathbf{v}_{\text{des}}(T) \end{bmatrix}, \quad (2.27)$$

for some $\boldsymbol{\nu} \in \mathbb{R}^9$. The necessary condition (2.26) supplies the additional $2n - 8$ equations to determine all the variables. A nonlinear equation solver can be used for this purpose. Alternatively, the first order optimality conditions of the augmented cost function

$$\bar{J}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T, \boldsymbol{\nu}) = \Psi^T \boldsymbol{\nu} + J(\mathbf{q}_f, \dot{\mathbf{q}}_f, T), \quad (2.28)$$

$$\begin{aligned} J(\mathbf{q}_f, \dot{\mathbf{q}}_f, T) &= \int_0^T \ddot{\mathbf{q}}^*(t)^T \ddot{\mathbf{q}}^*(t) dt, \\ &= \int_0^T (6\mathbf{a}_3 t + 2\mathbf{a}_2)^T (6\mathbf{a}_3 t + 2\mathbf{a}_2) dt, \\ &= 3T^3 \mathbf{a}_3^T \mathbf{a}_3 + 3T^2 \mathbf{a}_3^T \mathbf{a}_2 + T \mathbf{a}_2^T \mathbf{a}_2, \end{aligned} \quad (2.29)$$

directly satisfy (2.26) and the boundary equality constraints.

Parameter Optimization

The optimal trajectories are third order polynomials in joint-space for each degree of freedom of the robot, where the coefficients of the polynomials can be parameterized in terms of final joint positions \mathbf{q}_f , final joint velocities $\dot{\mathbf{q}}_f$ and hitting time T . That is, along with the hitting time T as a free parameter, the optimization problem is $2n + 1$ dimensional with nonlinear equality constraints. The integrand in (2.1) can be rewritten in terms of these free parameters and integrated over time as in (2.29) to form the following optimization problem

$$\min_{\mathbf{q}_f, \dot{\mathbf{q}}_f, T} 3T^3 \mathbf{a}_3^T \mathbf{a}_3 + 3T^2 \mathbf{a}_3^T \mathbf{a}_2 + T \mathbf{a}_2^T \mathbf{a}_2 \quad (2.30)$$

$$\text{s.t. } \mathbf{K}_p(\mathbf{q}_f) = \mathbf{b}(T), \quad (2.31)$$

$$\mathbf{K}_n(\mathbf{q}_f) = \mathbf{n}_{\text{des}}(T), \quad (2.32)$$

$$\mathbf{J}(\mathbf{q}_f) \dot{\mathbf{q}}_f = \mathbf{v}_{\text{des}}(T), \quad (2.33)$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_f \leq \mathbf{q}_{\max}, \quad (2.34)$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_{\text{ext}} \leq \mathbf{q}_{\max}. \quad (2.35)$$

The returning trajectories that bring the robot from striking joint positions \mathbf{q}_f to the fixed rest position \mathbf{q}_0 in joint space are also taken as third order polynomials

$$\mathbf{q}_{\text{return}}(t) = \tilde{\mathbf{a}}_3 t^3 + \tilde{\mathbf{a}}_2 t^2 + \dot{\mathbf{q}}_f t + \mathbf{q}_f, \quad (2.36)$$

for a fixed return time T_{rest} , $0 \leq t \leq T_{\text{rest}}$. The coefficients $\tilde{\mathbf{a}}_3$, $\tilde{\mathbf{a}}_2$ of (2.36) are as in (2.8) but with \mathbf{q}_0 , \mathbf{q}_f and $\dot{\mathbf{q}}_0$, $\dot{\mathbf{q}}_f$ reversed

$$\begin{aligned} \tilde{\mathbf{a}}_3 &= \frac{2}{T_{\text{rest}}^3} (\mathbf{q}_f - \mathbf{q}_0) + \frac{1}{T_{\text{rest}}^2} (\dot{\mathbf{q}}_f + \dot{\mathbf{q}}_0), \\ \tilde{\mathbf{a}}_2 &= \frac{3}{T_{\text{rest}}^2} (\mathbf{q}_0 - \mathbf{q}_f) - \frac{1}{T_{\text{rest}}} (\dot{\mathbf{q}}_0 + 2\dot{\mathbf{q}}_f). \end{aligned} \quad (2.37)$$

The optimization variables $\mathbf{q}_f, \dot{\mathbf{q}}_f$ fully parameterize the returning polynomials as well as the striking polynomials.

Joint Limit Satisfaction

Inspired by the simplicity of the Minimum Principle based solution, the same parameterization can be extended to the more realistic scenario where joint limits are included additionally as inequality constraints in the optimization. When optimizing (2.30) the final joint positions \mathbf{q}_f are enforced in (2.34) to respect the joint limits for each component. However, the whole trajectory, both the striking and returning segments, needs to respect the joint limits at all times. Third order polynomials can each have at most 2 extrema \mathbf{q}_{ext} in the interior of their domains, corresponding to the conditions

$$\dot{\mathbf{q}}_{\text{strike}}(t) = 3\mathbf{a}_3 t^2 + 2\mathbf{a}_2 t + \dot{\mathbf{q}}_0 = 0, \quad (2.38)$$

$$\dot{\mathbf{q}}_{\text{return}}(t) = 3\tilde{\mathbf{a}}_3 t^2 + 2\tilde{\mathbf{a}}_2 t + \dot{\mathbf{q}}_f = 0. \quad (2.39)$$

Algorithm 1 *Focused Player (FP)*

Require: $\mathbf{q}_0, \mathbf{b}_{\text{goal}}, T_{\text{land}}, T_{\text{pred}}, T_{\text{rest}}, N, \mathbf{R}$

```
1: Move to initial posture  $\mathbf{q}_0, \dot{\mathbf{q}}_0 = \mathbf{0}$ .
2: loop
3:   Query vision sys. for new observation  $\mathbf{b}_{\text{obs}}$ .
4:   Observe current state  $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$ .
5:   if  $N$  new ball observations  $\mathbf{b}_{\text{obs}}$  then
6:     Initialize EKF.
7:   end if
8:   if EKF is initialized and valid obs.  $\mathbf{b}_{\text{obs}}$  then
9:     Estimate position  $\mathbf{b}$  and vel.  $\dot{\mathbf{b}}$  with EKF.
10:    Predict  $\mathbf{b}(t)$  till horizon  $T_{\text{pred}}$ .
11:    Compute  $\mathbf{v}_{\text{des}}(t), \mathbf{n}_{\text{des}}(t)$  using racket model and boundary values  $\mathbf{b}_{\text{goal}}, T_{\text{land}}$ .
12:    Compute param.  $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$  from  $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$  using desired resting posture  $\mathbf{q}_0$ , time to return  $T_{\text{rest}}$ , weighting matrix  $\mathbf{R}$ , and task constraints  $\mathbf{b}(t), \mathbf{v}_{\text{des}}(t), \mathbf{n}_{\text{des}}(t)$ .
13:    Update strike and return trajectories  $\mathbf{q}_{\text{des}}(t) = \{\mathbf{q}_{\text{strike}}(t), \mathbf{q}_{\text{return}}(t)\}$ .
14:  end if
15:  Track  $\mathbf{q}_{\text{des}}(t)$  with Inv. Dyn.  $\boldsymbol{\tau} = \mathbf{f}(\mathbf{q}_{\text{des}}, \dot{\mathbf{q}}_{\text{des}}, \ddot{\mathbf{q}}_{\text{des}})$ .
16: end loop
```

Therefore, checking the joint extrema candidates \mathbf{q}_{ext} in (2.35) at times

$$\begin{aligned} v_j^{1,2} &= \frac{-a_{2,j} \pm \sqrt{a_{2,j}^2 - 3a_{3,j}\dot{q}_{0,j}}}{3a_{3,j}}, \\ v_j^{3,4} &= \frac{-\tilde{a}_{2,j} \pm \sqrt{\tilde{a}_{2,j}^2 - 3\tilde{a}_{3,j}\dot{q}_{f,j}}}{3\tilde{a}_{3,j}}, \end{aligned} \quad (2.40)$$

for each $j = 1, \dots, n$ makes sure that the joint limits are satisfied both for the striking trajectory (at times $v_j^{1,2}$) and for the returning trajectory (at times $v_j^{3,4}$). These candidate times are fully parameterized by the optimization variables since the coefficients $\mathbf{a}_3, \mathbf{a}_2$ (2.8) and $\tilde{\mathbf{a}}_3, \tilde{\mathbf{a}}_2$ (2.36) appearing in (2.40) can be determined whenever $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ are computed. The values $v^{1,2}$ are clamped to the interval $[0, T_{\text{pred}}]$ and $v^{3,4}$ to $[0, T_{\text{rest}}]$ if they are imaginary or outside their corresponding intervals.

Online Trajectory Generation

Using a constrained nonlinear optimizer, the algorithm can be run online whenever there are enough ball samples $N = 12$ available to estimate the incoming ball state and spin reliably. After computing an initial striking trajectory and starting to move, the trajectories can be corrected online whenever new ball samples are available.

The full trajectory generation framework and the resulting table tennis player *Focused Player* (FP) is summarized in Algorithm 5. After bringing the robot to a desired initial posture \mathbf{q}_0 , the vision system is queried (line 3) for new reliable ball observations. The Extended Kalman Filter (EKF) is initialized (line 6) using the first $N = 12$ ball positions. EKF then updates the ball

state whenever new ball observations \mathbf{b}_{obs} are available. The ball state is used to predict every $dt = 2$ ms, a discrete set of ball positions and velocities along the future ball path $\mathbf{b}(t)$, up to a horizon of $T_{\text{pred}} = 1.0$ s. Desired racket parameters are then computed (line 11) for each $t = dt, \dots, T_{\text{pred}}$, before the optimization for the robot joint movements is launched. With an optimized implementation, the optimization (line 12) takes on average 25 ms to find the trajectory parameters for the Barrett WAM. The optimized implementation thus makes it possible to implement the approach online in the robot table tennis setup. See section 4.4 for the implementation details.

The desired landing position \mathbf{b}_{goal} and T_{land} are important free parameters of the algorithm, that can possibly be set by a higher-level strategy. For instance, a fast playing robot would prefer to set T_{land} rather low, and given an opponent state, a robot that wants to score a point could profit from adapting the desired landing position as well. We discuss in the Experiment section the effects of changing these parameters for the overall returning accuracy of the Barrett WAM.

The optimization takes place online (line 12) whenever new reliable ball observations and robot joint sensor recordings \mathbf{q}_{cur} are available. The desired robot movement can be updated to accommodate for modeling and control errors. Feasible striking and return trajectories are then formed or updated (line 13), which are executed with an existing inverse dynamics controller. In actual table tennis experiments, we apply high gain PD-control in addition to inverse dynamics (computed torque). See Section 4.4 for more details of how the algorithm runs online in actual robot table tennis experiments.

For simplicity we have not introduced a weighting matrix in (2.30). We include in Algorithm 5 an arbitrary positive definite weighting matrix \mathbf{R} , which can be used to emphasize for each degree of freedom the difficulty of accelerating that particular joint.

2.5 The Defensive Player

The optimal control problem introduced in (2.1) can be solved directly without the additional Cartesian constraints considered in the previous section. As opposed to fixing a desired landing point and a desired landing time to satisfy the requirements of a higher-level strategy, there can be times during table tennis where it is much more important to safely return the ball. A *defensive* table tennis player could relax the previously imposed racket constraints (2.18) – (2.20) by requiring only that the task constraints (2.2) – (2.4) are satisfied. See Figure 2.5 for an illustration.

2.5.1 Table Tennis Task Constraints

The indoor environment that is modeled contains a standard ping pong table with coordinates

$$\mathcal{T} = \{(x, y, z_T) \in \mathbb{R}^3 \mid -\frac{w_T}{2} \leq x \leq \frac{w_T}{2}, \quad y_{\text{edge}} - l_T \leq y \leq y_{\text{edge}}\}, \quad (2.41)$$

where the origin is placed at the robot base. The table with width $w_T = 152$ cm and length $l_T = 276$ cm is approximately at $z_T = -0.89$ cm height and placed $|y_{\text{edge}}| = 115$ cm away from the robot base, see Figure 4.1. The racket and the table tennis ball have a radius of $r_R \approx 7.6$ cm and $r_B = 2$ cm, respectively. The condition for successful landing can be put succinctly as follows: the ball after the hit has to pass over the net, below the wall and land on the opponents court. See Figure 2.3 for an illustration.

Hitting Constraint

All possible impacts of the racket with the ball at time T are captured by the *hitting set* \mathcal{H}

$$\begin{aligned}\mathcal{H} = \{ & (T, \mathbf{r}(T), \mathbf{n}(T)) \in \mathbb{R}^7 \mid T \geq 0, \\ & 0 \leq \mathbf{n}(T)^\top (\mathbf{b}(T) - \mathbf{r}(T)) \leq r_B, \\ & \|\mathbf{P}_n^\perp(T)(\mathbf{b}(T) - \mathbf{r}(T))\| \leq r_R\},\end{aligned}\quad (2.42)$$

where $\mathbf{P}_n^\perp(t) = \mathbf{I} - \mathbf{n}(t)\mathbf{n}^\top(t)$ is the projection matrix onto the racket plane. The hitting function

$$\Psi_{\text{hit}}(\mathbf{q}(T), T) = \begin{pmatrix} T \\ \mathbf{K}_p(\mathbf{q}(T)) \\ \mathbf{K}_n(\mathbf{q}(T)) \end{pmatrix} \quad (2.43)$$

enforces the kinematic constraints for hitting when $\Psi_{\text{hit}}(\mathbf{q}(T), T) \in \mathcal{H}$.

Net Constraint

When crossing the net at time t , the ball should be above the net height z_{net} and below the wall z_{wall} , that is, $(t, \mathbf{b}(t))$ should belong to the set

$$\begin{aligned}\mathcal{N} = \{ & (T_{\text{net}}, \mathbf{b}(T_{\text{net}})) \in \mathbb{R}^4 \mid T_{\text{net}} > 0, \\ & b_y(T_{\text{net}}) = y_{\text{net}} := y_{\text{edge}} - \frac{l_T}{2}, \\ & z_{\text{net}} \leq b_z(T_{\text{net}}) \leq z_{\text{wall}}\}.\end{aligned}\quad (2.44)$$

The net hitting time T_{net} is calculated by using the ball prediction functions,

$$T_{\text{net}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \{t \mid b_y(t) = y_{\text{net}}\}. \quad (2.45)$$

The net function Ψ_{net} that predicts the future ball position on the vertical net plane

$$\Psi_{\text{net}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \begin{pmatrix} T_{\text{net}} \\ b_x(T_{\text{net}}) \\ y_{\text{net}} \\ b_z(T_{\text{net}}) \end{pmatrix} \quad (2.46)$$

is then the composition of a ball-racket contact model with the ball flight model.

Landing Constraint

The desired condition for landing afterwards in the opponents court will then be

$$\begin{aligned}\mathcal{L} = \{ & (T_{\text{land}}, \mathbf{b}(T_{\text{land}})) \in \mathbb{R}^4 \mid T_{\text{land}} > T_{\text{net}}, \\ & b_z(T_{\text{land}}) = z_T + r_B, \\ & -\frac{w_T}{2} \leq b_x(T_{\text{land}}) \leq \frac{w_T}{2}, \\ & y_{\text{net}} - \frac{l_T}{2} \leq b_y(T_{\text{land}}) \leq y_{\text{net}} \}.\end{aligned}\quad (2.47)$$

The landing time T_{land} at which the ball hits the horizontal table plane, is found using the ball prediction functions

$$T_{\text{land}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \{t > T_{\text{net}} \mid b_z(t) = z_T + r_B\}. \quad (2.48)$$

The landing function Ψ_{land} that predicts the future ball position on the horizontal table plane,

$$\Psi_{\text{land}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) = \begin{pmatrix} T_{\text{land}} \\ b_x(T_{\text{land}}) \\ b_y(T_{\text{land}}) \\ z_T + r_B \end{pmatrix}, \quad (2.49)$$

is, as before, the composition of a ball-racket contact model with the ball flight model.

2.5.2 Nonlinear Constrained Optimization

We briefly show here that the solution $\mathbf{q}(t)$ to the original optimal control problem posed in (2.1) – (2.6), with additional penalties for landing and hitting, is a third order polynomial for each degree of freedom, $i = 1, \dots, n$. The penalties for landing and hitting can be grouped together as ϕ_{pen} , where

$$\begin{aligned}\phi_{\text{pen}} &= \alpha_{\text{hit}} \phi_{\text{hit}}(\mathbf{q}_f, T) + \alpha_{\text{land}} \phi_{\text{land}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T), \\ \phi_{\text{hit}} &= (\mathbf{b}(T) - \mathbf{r}(T))^T \mathbf{P}_n^\perp(T) (\mathbf{b}(T) - \mathbf{r}(T)), \\ \phi_{\text{land}} &= (\mathbf{b}(T_{\text{land}}) - \mathbf{b}_{\text{goal}})^T (\mathbf{b}(T_{\text{land}}) - \mathbf{b}_{\text{goal}}).\end{aligned}\quad (2.50)$$

with tunable weights α_{hit} and α_{land} .

Derivation from Minimum Principle

The same derivation in section 2.4.2 applies for the Hamiltonian and the momenta. Instead of the boundary equality constraints we get the more general inequality constraints at striking time

$$\begin{aligned}-\mathcal{H}(T) &= \frac{\partial \Phi}{\partial T}, \\ \lambda^*(T) &= \frac{\partial \Phi}{\partial \mathbf{q}(T)}, \\ \mu^*(T) &= \frac{\partial \Phi}{\partial \dot{\mathbf{q}}(T)},\end{aligned}\quad (2.51)$$

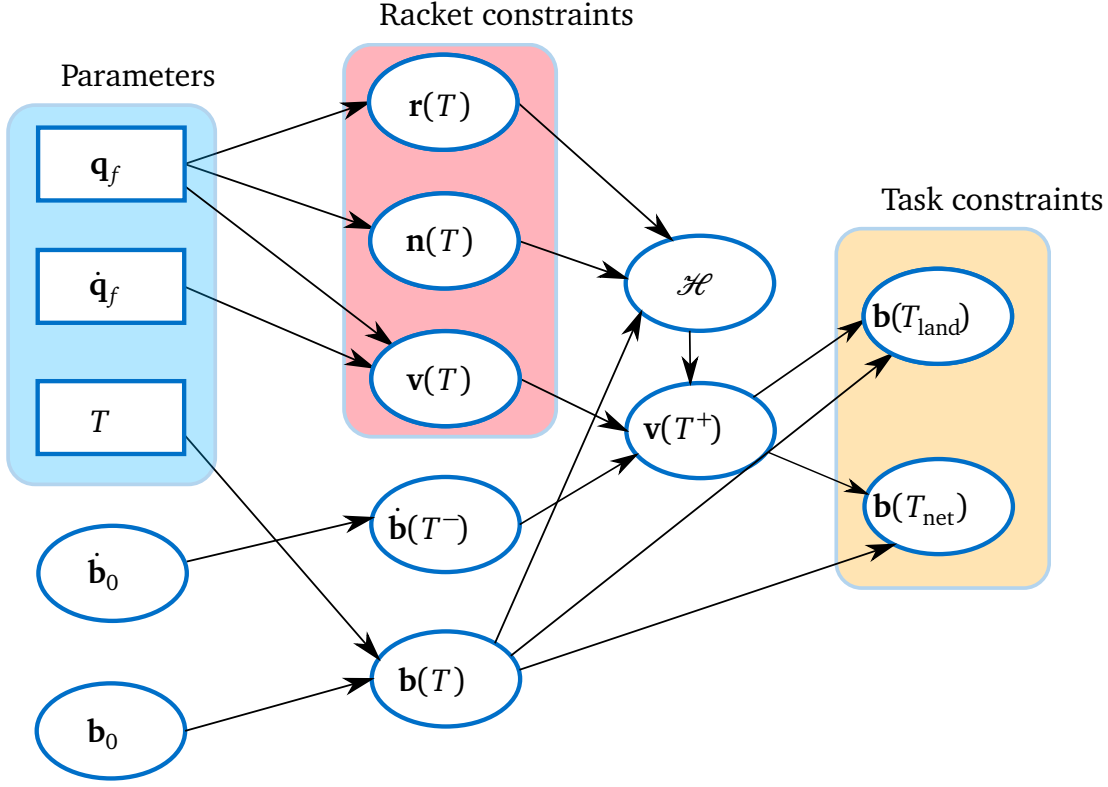


Figure 2.5: Graphical representation of table tennis interactions. The hybrid system for the table tennis ball is described by the flight dynamics, governed by a set of differential equations, as well as a discrete hitting event \mathcal{H} that changes the ball velocity from $\dot{\mathbf{b}}(T^-)$ to $\dot{\mathbf{b}}(T^+)$ at the hitting time T . The control variables for the reduced optimization problem are located in the light blue rectangle. Racket constraints that are enforced by *Focused Player* to land the ball to a fixed location are indicated in the red rectangle. *Defensive Player* on the other hand, directly enforces the task (landing and net) constraints, located in the orange rectangle, without additional constraints. By additionally checking for the hitting condition \mathcal{H} in the optimization, this problem can be cast as a (standard) continuous optimal control problem, where the decision variables \mathbf{q}_f , $\dot{\mathbf{q}}_f$ and T continuously affect the outgoing ball velocity, the ball net and landing positions, through the repeated application of the flight model (2.9) and the contact model (2.14).

where the generalized boundary cost is

$$\Phi(\mathbf{q}, \dot{\mathbf{q}}, T, \boldsymbol{\nu}) = \phi_{\text{pen}} + \boldsymbol{\nu}^T \boldsymbol{\Psi}_{\text{strike}} \quad (2.52)$$

for some Lagrange multipliers $\boldsymbol{\nu} \in \mathbb{R}^{13}$ and $\boldsymbol{\Psi}_{\text{strike}} \leq \mathbf{0}$ representing the hitting, landing and net inequality constraints (2.2) – (2.4). The conditions (2.51) along with primal feasibility, complementary slackness and dual feasibility conditions

$$\begin{aligned} \boldsymbol{\Psi}_{\text{strike}}(\mathbf{q}(T), \dot{\mathbf{q}}(T), T) &\leq \mathbf{0}, \\ \boldsymbol{\Psi}_{\text{strike}}^T \boldsymbol{\nu} &= 0, \\ \boldsymbol{\nu} &\geq \mathbf{0}, \end{aligned} \quad (2.53)$$

respectively, supply the additional equations to determine all the variables. The first order optimality conditions of the augmented cost function

$$\tilde{J}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T, \boldsymbol{\nu}) = \Phi(\mathbf{q}_f, \dot{\mathbf{q}}_f, T, \boldsymbol{\nu}) + J(\mathbf{q}_f, \dot{\mathbf{q}}_f, T), \quad (2.54)$$

directly satisfy (2.51) and the associated boundary inequality constraints.

Parameter Optimization

With the same parameterization as in *Focused Player* (FP), the cost functional is extended with an additional penalty term $\phi_{\text{pen}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T)$ while enforcing the more general inequality constraints

$$\min_{\mathbf{q}_f, \dot{\mathbf{q}}_f, T} 3T^3 \mathbf{a}_3^T \mathbf{a}_3 + 3T^2 \mathbf{a}_3^T \mathbf{a}_2 + T \mathbf{a}_2^T \mathbf{a}_2 + \phi_{\text{pen}} \quad (2.55)$$

$$\text{s.t. } \boldsymbol{\Psi}_{\text{strike}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T) \leq \mathbf{0}, \quad (2.56)$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_f \leq \mathbf{q}_{\max}, \quad (2.57)$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_{\text{ext}} \leq \mathbf{q}_{\max}. \quad (2.58)$$

The components $\phi_{\text{land}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T)$ and $\phi_{\text{hit}}(\mathbf{q}_f, T)$ of ϕ_{pen} impose additional penalties on the hitting joint positions and velocities. Unlike the FP, the joint extrema \mathbf{q}_{ext} are only checked for the striking trajectory, as the returning trajectory is the result of an additional optimization.

Resting State Optimization.

For the DP, we additionally consider a resting posture optimization to find a more *defensive* posture for the robot. By finding a joint resting state \mathbf{q}_0 that minimizes both the distance from the hitting state \mathbf{q}_f and the squared Frobenius norm of the Jacobian at the resting state

$$\min_{\mathbf{q}_0, t} (\mathbf{q}_0 - \mathbf{q}_f)^T (\mathbf{q}_0 - \mathbf{q}_f) + \|\mathbf{J}(\mathbf{q}_0)\|_F^2 \quad (2.59)$$

$$\text{s.t. } 0 \leq t \leq T_{\text{pred}}, \quad (2.60)$$

$$\mathbf{K}_p(\mathbf{q}_0) = \mathbf{b}(t), \quad (2.61)$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_0 \leq \mathbf{q}_{\max}, \quad (2.62)$$

$$\mathbf{q}_{\min} \leq \mathbf{q}_{\text{ext}} \leq \mathbf{q}_{\max}, \quad (2.63)$$

Algorithm 2 *Defensive Player (DP)*

```
Require:  $\mathbf{q}_0, \mathbf{R}, N, T_{\text{pred}}, \alpha_{\text{hit}}, \alpha_{\text{land}}$ 
1: Wait at initial posture  $\mathbf{q}_0$ .
2: loop
3:   Query vision sys. for new observation  $\mathbf{b}_{\text{obs}}$ .
4:   Observe current state  $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$ .
5:   if  $N$  new ball observations  $\mathbf{b}_{\text{obs}}$  then
6:     Initialize EKF.
7:   end if
8:   if EKF is initialized and valid obs.  $\mathbf{b}_{\text{obs}}$  then
9:     Estimate position  $\mathbf{b}$  and vel.  $\dot{\mathbf{b}}$  with EKF.
10:    Predict  $\mathbf{b}(t)$  till horizon  $T_{\text{pred}}$ .
11:    Compute  $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$  from  $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$  using  $\mathbf{b}(t)$  and the weights  $\mathbf{R}, \alpha_{\text{hit}}, \alpha_{\text{land}}$ .
12:    Update  $\mathbf{q}_0$  using  $\mathbf{q}_f, \mathbf{b}(t)$ 
13:    Update strike and return trajectories  $\mathbf{q}_{\text{des}}(t) = \{\mathbf{q}_{\text{strike}}(t), \mathbf{q}_{\text{return}}(t)\}$ .
14:   end if
15:   Track  $\mathbf{q}_{\text{des}}(t)$  with Inv. Dyn.  $\tau = \mathbf{f}(\mathbf{q}_{\text{des}}, \dot{\mathbf{q}}_{\text{des}}, \ddot{\mathbf{q}}_{\text{des}})$ .
16: end loop
```

such that the Cartesian resting state intersects the ball path for some t , $0 \leq t \leq T_{\text{pred}}$, we can minimize the amount of movement necessary to return the next incoming ball. The feasibility of the third order polynomials that goes from hitting state $\mathbf{q}_f, \dot{\mathbf{q}}_f$ to $\mathbf{q}_0, \dot{\mathbf{q}}_0 = \mathbf{0}$ is ensured by including the joint extrema candidates throughout the returning trajectory in (2.63). Including the Frobenius norm of the Jacobian in the cost function makes sure that the striking trajectories will be easy to generate (i.e., have low accelerations) for the next predicted ball trajectories near the last ball trajectory.

Online Trajectory Generation

The resulting table tennis player is summarized in pseudocode format in Algorithm 6. As in Algorithm 5, the algorithm can be run online whenever there are enough ball samples $N = 12$ available to estimate the incoming ball reliably. The ball state is used, as before, to predict every $dt = 2$ ms, a discrete set of ball positions and velocities along the future ball path $\mathbf{b}(t)$, up to a horizon of $T_{\text{pred}} = 1.0$ s. The optimization for the striking trajectory (line 11) is then launched, which takes on average 25 ms to find a local optimum for the Barrett WAM. Good initialization and an optimized implementation make it possible to implement the approach online in our robot table tennis setup. See section 4.4 for the implementation details.

After computing an initial striking trajectory and starting to move, the trajectories can be corrected online (line 11-13) whenever there are new ball samples available. Compared to *Focused Player*, the gained flexibility due to relaxed constraints is increased with the addition of the resting posture optimization (line 12) that reduces the accelerations of the next hitting movements for similar incoming balls.

Similar to Algorithm 5, we include in Algorithm 6 an arbitrary positive definite weighting matrix \mathbf{R} , which together with the task weights α_{hit} and α_{land} , adjusts the weight of a particular degree of freedom's accelerations in calculating the total cost (2.55).

2.6 Experiments & Evaluations

In this section, we will evaluate the performance of the online trajectory generation algorithms in simulation and in real robot table tennis experiments. We first start by comparing the ball returning performance of *Focused Player* (FP) in simulation against the virtual hitting plane (VHP) method.

2.6.1 Simulation Studies

In simulation the performance of the new table tennis players can be extensively evaluated without robot control or ball prediction errors. We will make controlled experiments to first show that the player FP outperforms the VHP based player, and can generate striking trajectories more robustly.

Virtual Hitting Plane Method

The VHP method that we implement is a close variant of [9]. In this approach, the specification of the VHP (see Figure 2.6) fixes the hitting time T as well as the hitting point $\mathbf{b}(T)$ for the racket trajectory. The remaining parameters, the desired racket velocity $\mathbf{v}_{\text{des}}(T)$ and the desired racket normal at hitting time $\mathbf{n}_{\text{des}}(T)$ are calculated by inverting the models (2.9) and (2.14) at the hitting time T .

To run inverse kinematics (IK) on the desired hitting point, one needs to additionally specify a desired racket slide [36]. An easy and convenient way to generate a desired racket slide at hitting time is to rotate the initial racket slide until the initial racket normal aligns with the final desired racket normal. This procedure determines the full orientation of the final robot posture at hitting time.

After specifying the orientation of the robot at hitting time, Jacobian pseudoinverse based IK can be run to determine the final joint positions. IK typically takes less than 2 ms to converge to \mathbf{q}_f . Final joint velocities are then found by using the Jacobian at hitting time $\mathbf{J}(\mathbf{q}_f)$ and the desired racket velocities

$$\dot{\mathbf{q}}_f = \mathbf{J}^\dagger(\mathbf{q}_f) \mathbf{v}_{\text{des}}(T). \quad (2.64)$$

The computed parameters $\mathbf{q}_f, \dot{\mathbf{q}}_f$ along with the fixed hitting time T fully determine a third degree polynomial in joint space for each degree of freedom of the robot $i = 1, \dots, n$. The joint limitations are then checked and the procedure is repeated if the accelerations are too high. When the ball is coming close to the robot's initial posture \mathbf{q}_0 , this complicated IK procedure results in feasible trajectories if the VHP is chosen appropriately. However, it is rather inflexible and can easily fail to find good configurations.

Comparison with the VHP Method

To make a fair comparison between the VHP approach and our algorithm FP, in our simulation environment¹ the initial ball state variance is fixed such that most balls end up close to the initial

¹ Code for the simulation platform as well as a video showing some example trajectories is available in the GitHub repository: <https://github.com/RobotLearning/traj-gen-and-tracking.git>

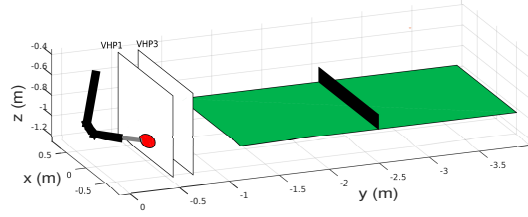


Figure 2.6: For simulating the performance of the virtual hitting plane (VHP) based method in a fair way, the results are averaged over four different VHP locations. The first and third plane locations are shown in the figure. Out of 50 balls each, the VHP at $y = -0.7$, $y = -0.6$, $y = -0.5$, $y = -0.4$ return 31, 37, 28, 29 balls respectively.

robot posture. This ensures that a fair evaluation between the two algorithms can be given. Both methods filter the incoming stream of ball position estimates using the same Extended Kalman Filter (EKF) and equally start moving whenever $N \approx 12$ balls are detected.

Evaluations are summarized in Table 2.2. A total of 200 balls are launched towards the robot in single-ball solo trials from varying initial positions and velocities, $\mathbf{b}_{\text{init}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{init}}, \sigma_{\text{init}}^2 \mathbf{I})$. The initial ball mean positions are fixed on the left corner of the opponent's court and the initial covariance matrix is diagonal with a standard deviation of $\sigma_{\text{init}} = 0.1$. Some balls are illegal, for example they might not bounce on the robot's court. Such balls are detected with our ball prediction models and they are not considered for strike generation. They are marked as *Not Valid* in Table 2.2.

Comparing with the VHP method, it can be seen that FP is able to return more balls to the other side, with 26 more balls returned to the opponent's court. One of the main reasons for this increase in performance is the fixed location of the VHP. Depending on the incoming ball velocity, trajectories generated using a fixed VHP can result in joint limit violations or infeasible solutions. A second reason is the explicit incorporation of joint limits both for the striking trajectory and the returning trajectory in the optimization problem. Both cases are included as *Infeasible* in Table 2.2. See Figure 2.6 for an illustration. Out of 50 balls each, the VHP methods with the planes fixed at $y = -0.7$, $y = -0.6$, $y = -0.5$, $y = -0.4$ locations return 31, 37, 28, 29 balls respectively. For this particular ball distribution, the plane at $y = -0.6$ seems to be the most robust option. In terms of landing point accuracy, both methods achieve a roughly isotropic Gaussian distribution with variance $\sigma^2 = 0.15m$, with varying desired landing points \mathbf{b}_{goal} as the mean.

Lookup Table

A naive implementation of FP in MATLAB using Sequential Quadratic Programming (SQP), takes about two seconds on our system on average. [1] proposed a lookup table as a remedy to replace

Table 2.2: Results comparing FP and VHP

	Returns	Not Valid	Infeasible	Miss/Out
FP	151	24	19	6
VHP	125	24	45	6

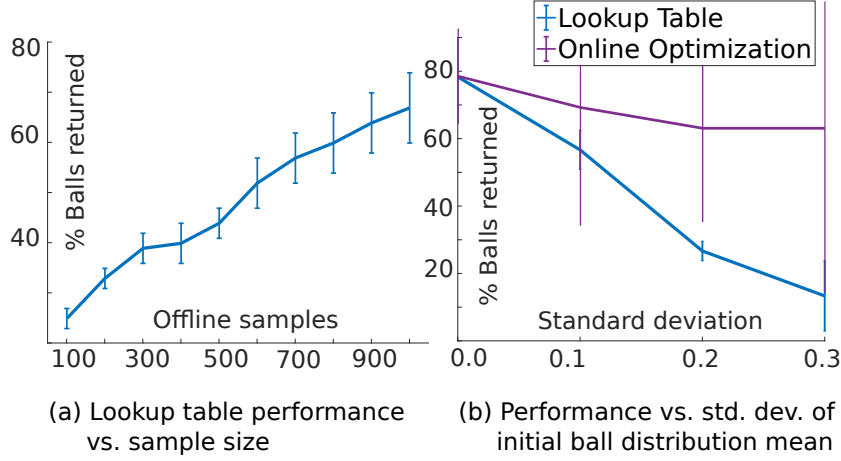


Figure 2.7: As an alternative to computing the trajectory parameters online, [1] proposed a lookup table to generate trajectories. Performance of the trajectory generation framework using a lookup table is shown in blue. Results are averaged over 5 different runs. As the number of stored lookup table samples increase, the performance approaches that of the online trajectory generation in (a). However, as shown in (b), even the performance of a lookup table with 4000 entries degrades quickly whenever ball position and velocity estimates are not close to the values stored in the lookup table.

online optimization. Whenever a strike computed offline is successful in returning the ball in simulation, ball positions, velocities at the start of the movement and the optimized parameters $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ can be stored in a lookup table. One can then at runtime simply lookup the optimized parameters that have the closest stored ball position and velocity estimates to new ball estimates. The performance of this lookup table based approach is evaluated in Figure 2.7. The same initial ball distribution with the same $\mu_{\text{init}}, \sigma_{\text{init}}$ values is used as before. As the number of lookup table samples increase, the percentage of incoming balls returned successfully approaches that of the online trajectory generation.

The simple lookup table approach that is employed here corresponds to k-nearest neighbor interpolation with $k = 1$. Machine learning based methods that regress between lookup table entries using more sophisticated approaches are discussed extensively in, for example, [37].

Online Trajectory Generation

The lookup table proposed above is based on a fixed initial posture \mathbf{q}_0 while the robot is at rest, i.e., $\dot{\mathbf{q}}_0 = \mathbf{0}$. Its performance degrades whenever filtered ball positions \mathbf{b}_0 and velocities $\dot{\mathbf{b}}_0$ are not close to the values stored in the lookup table, or when the initial posture is different. See Figure 2.7 for the decrease in performance of a lookup table with 4000 entries, as the mean of the initial ball distribution, $\mu \sim \mathcal{N}(\mu_{\text{init}}, \sigma^2 \mathbf{I})$, is randomized with increasing variances σ^2 .

To overcome the shortcomings of a lookup-table based approach, we implemented the optimizations in C++ with an interface to the simulation environment SL [38]². SL is also our real-time interface to the robot and runs at a frequency of 500 Hz, terminating any processes

² C++ code for the online optimization run in the real-time simulation platform SL can be found in: <https://github.com/RobotLearning/table-tennis.git>.

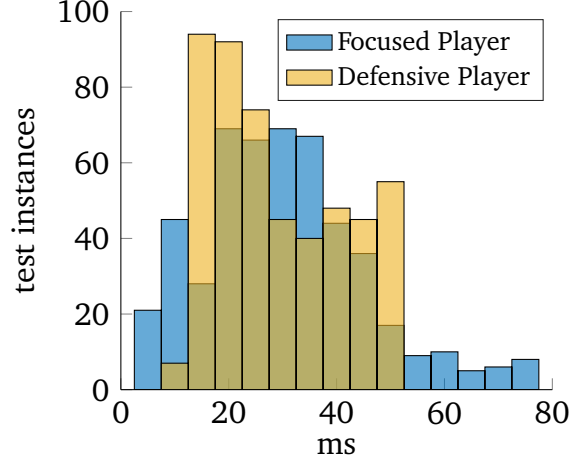


Figure 2.8: Histogram of the runtime distributions of the two players, evaluated over 500 random test instances. Both algorithms FP and DP have an average runtime of about 25 ms, but for FP, the distribution is wider. For evaluating DP we have regressed on a lookup table using k-nearest-neighbor (kNN) regression. Without kNN, the runtime distribution for DP concentrates sharply around 50 ms.

that do not finish within 2 milliseconds. It is mainly responsible for running the inverse dynamics and feedback control loop computations. To run the optimization online, a thread separate from the one running the inverse dynamics is launched, whenever there are reliable ball observations available and another thread is not running.

We use the NLOpt library [39] to run both optimizations. For the *Focused Player* (FP), we found that among the nonlinear constrained optimization routines, only COBYLA respects the equality constraints given in (2.18) – (2.20). The algorithm COBYLA is a simplex method implemented in NLOpt that uses direct search with linear approximations [40]. Gradients of the cost function (2.30) can be easily calculated and fed to a gradient based solver, but this direct search routine takes only about 25 ms to converge, i.e., about the same frequency as the incoming ball observations.

Before the optimization for robot striking movements take place, the desired outgoing ball velocities and the corresponding racket parameters $\mathbf{v}_{\text{des}}(t)$, $\mathbf{n}_{\text{des}}(t)$ necessary to enforce the equality constraints (2.18) – (2.20) are predicted every 2 ms for $0 \leq t \leq T_{\text{pred}}$. The prediction horizon $T_{\text{pred}} = 1.0$ is more than enough, given the speed of the balls, for the balls to pass the robot workspace. Racket computations take on average 1 – 1.5 ms for the whole sequence of predicted ball states. The discretization of 2 ms is natural, since the robot control runs at 500 Hz. During the optimization for a continuous T , the corresponding racket variables (racket desired positions, velocities and normals) are interpolated linearly between the discrete predicted values.

For the *Defensive Player* (DP), the Augmented Lagrangian (AUGLAG) method is used to convert the problem to an unconstrained optimization problem, which is then solved with a Quasi-Newton algorithm. In this case, only incoming ball positions and velocities are predicted, again discretized over 2 milliseconds.

Good initialization does not always guarantee faster convergence, but it can help escape bad local minima of the cost functions. The optimization parameters for FP are first initialized to the resting posture, $\mathbf{q}_f = \mathbf{q}_0$, $\dot{\mathbf{q}}_f = \mathbf{0}$ and $T = 0.5\text{s}$. Whenever the robot is already moving

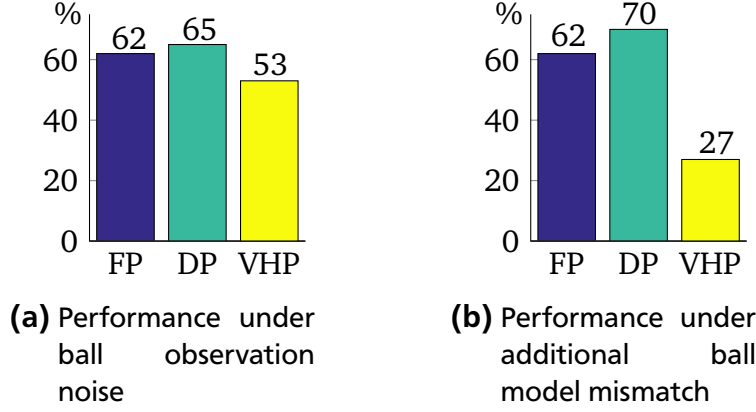


Figure 2.9: Simulation results comparing the return accuracy of three table tennis players. In (a), ball positions are observed with Gaussian white noise. In (b), there is an additional mismatch due to unknown topspin. Out of 200 balls, 14 and 12 incoming balls did not bounce legally and were not considered for trajectory generation, respectively. The other balls that were not counted as returns were either missed, or did not land legally on the opponent’s court.

and corrections are being computed, the parameters are initialized to their previously computed values. For DP, we initialize by regressing on a lookup table using k-nearest-neighbor (kNN) regression, with $k = 5$, to speedup the optimization process. Figure 2.8 shows the runtime distributions of the two algorithms over 500 test instances. In each trial, the ball is launched from different sides with the same distribution as described before, and the robot initial posture is also chosen randomly. Both algorithms FP and DP have approximately an average runtime of 25 ms, but for FP, the distribution is wider. Without regressing on a lookup table, the distribution for DP concentrates sharply around 50 ms.

Online Corrections

In order to show the performance increase due to online corrections, we first put Gaussian white noise with $\sigma = 0.02$ m standard deviation on the ball observations and apply Extended Kalman Filter (EKF). As the ball approaches, the robot gets increasingly better estimates of the ball state. Our real-time simulator runs at 500 Hz, while the ball observation is limited to 60 Hz, limiting the frequency of online corrections. The results are summarized in Figure 2.9a, averaged over three different ballgun locations: left, center, and right. The initial pose of the robot is placed opposite accordingly, i.e. on the right side of the table if the ball is coming from the left. Out of 200 balls, 14 incoming balls did not bounce legally and were not considered for trajectory generation. The other balls that were not returned successfully were either missed, or did not land legally on the opponent’s court. The online optimization is started whenever there are $N = 12$ ball samples available. This is enough to ensure that the estimated ball velocities will not cause robot movements that are far off from the ball. The solver then continues at a rate of 25 Hz until the ball appears behind the racket centre, i.e., $b_y > r_y$. Any ball that suddenly appears on the opponent’s court causes the Kalman Filter to reset, reinitialized with that ball observation as the initial mean and with a high variance.

The players that generate trajectories only once are able to return only very few balls (10 on average) in this mode of evaluation. Correcting with the VHP method improves the returning performance significantly. However, balls that are not feasible for the robot in the Virtual Hitting Plane intersection cannot be returned at all with this player. *Focused Player* (FP) and *Defensive Player* (DP) on the other hand, can find and generate feasible movements more flexibly. FP and VHP methods both return the balls with a roughly isotropic Gaussian distribution around the center of the table, with a variance of $\sigma_{\text{land}}^2 = 0.3m$, while the ball landing positions \mathbf{b}_{land} of DP follow roughly a uniform distribution.

As an additional challenge, we also consider the model mismatch case where there is a very high topspin on the ball, around 3000 revolutions per minute (rpm). EKF assumes a nonspinning model for the ball, i.e., $C_L = 0$. The solver is run with an increased rate of 50 Hz to be able to return the balls. The players that generate trajectories only once are not able to return any balls in this aggressive mode of evaluation. The results are summarized in Table 2.9b. 12 incoming balls out of 200 did not bounce legally and were not considered for trajectory generation. As in the previous experiment, FP and DP correct the trajectories more easily and overall return more balls. The advantage of DP over FP in this setting is due to the more flexible returning criterion, as the resting state optimization was not applied. In terms of landing point accuracy, both algorithms have similar ball landing distributions as before, but the means have an offset of $0.3m$ closer to the other side of the table. The offset is due to the fact that the racket computations for FP and the landing point calculations for DP in this case do not assume a spin model for the ball.

2.6.2 Real Robot Table Tennis

In this section we describe and discuss our experiments on the robotic table tennis setup, see Figure 4.1.

Description of the Setup

Our robot is a seven degree of freedom Barrett WAM arm that can easily reach $10g \text{ m/s}^2$ accelerations. It is torque-controlled and cable-driven. A standard size racket (7.6 cm radius) is attached to the end-effector. The vision system tracks the balls at a rate of 60 Hz and consists of four cameras on the corners on the ceiling. See [41] for platform details. The table and the tennis balls are standard sized, the balls have a radius of 2 cm, the table geometry is approximately $276 \times 152 \times 76 \text{ cm}$.

In the robot experiments, we use a ball-launcher (see Figure 4.1) to throw balls to the the robot, approximately once every 2 – 3 seconds. The balls generally come with a high variance, especially the velocities are quite unpredictable even without oscillating the ball-launcher. The robot base is at a distance of 115 cm to the end of the table and 95 cm above the table. Robot base is centered with respect to the table in the x direction, see Figure 4.1.

Estimation and Outlier Detection

The ball detection algorithm detects the center of mass of the orange balls from each image separately and fuses them together to form two ball position estimates. These are then filtered with an Extended Kalman Filter (EKF) to estimate ball position and velocity.

After the ball-launcher shoots a ball, $N = 12$ ball observations are used to initialize the Kalman Filter state and launch the trajectory generation process, see Algorithms 5 and 6. Balls that suddenly appear on the opponent's court after disappearing for more than 0.5 seconds from the cameras cause the Kalman Filter to reset. The filter state and the ball spin (assumed constant throughout motion) are then estimated together with a truncated Newton's method³ using $N = 12$ ball samples. We have experimentally confirmed the value of N to be a good compromise between ball estimation accuracy (which requires waiting) and moving early (which can reduce the accelerations).

Online Correction of Computed Trajectories

Since the ball is moving at fast speeds, our online trajectory generation algorithms need to be on the order of tens of milliseconds, in order to reliably intercept the incoming ball. The optimizers take on average 25 ms to converge, and they can be re-run in the real-time platform whenever there are new reliable ball observations \mathbf{b}_{obs} . Before launching the trajectory optimizers, the path of the ball is predicted each time for $T_{\text{pred}} = 1.0$ seconds and the algorithms are initialized with current joint state estimates $\mathbf{q}_{\text{cur}}, \dot{\mathbf{q}}_{\text{cur}}$. The optimizations are performed in the same way as described in the previous sections. The only difference is that during the optimization process, the ball is approaching the robot, hence we subtract the optimization time T_{run} from the computed desired hitting time T before generating the hitting trajectory, i.e., $T \leftarrow T - T_{\text{run}}$.

During the correction process, we make sure that the updates are always incremental and feasible. For completeness, we list here our software checks. We make sure that:

1. At least $N = 12$ reliable ball observations are available. This typically happens before the balls pass the net.
2. The new ball estimate is not too far off from the previous estimates.
3. Our previous optimization thread has terminated before another one is launched.
4. The resulting Cartesian trajectory intersects with the ball and all the task constraints are satisfied.
5. The corrections are never excessive, i.e., the acceleration and joint limits (2.34) and (2.35) are always respected.
6. The ball estimate appears to be in front of the robot, i.e., $b_y < r_y$.

If any of these conditions are violated, then the trajectories are not updated, and the previous striking trajectory is followed without interruption. The balls come with a high variance in position and especially in velocity. Typical incoming ball velocities imparted by the ball-launcher are around 4 – 6 m/s range in the y-direction, which implies that in practice there can be a

³ The optimization is launched on another thread using the *TNEWTON* algorithm in NLOpt [39].

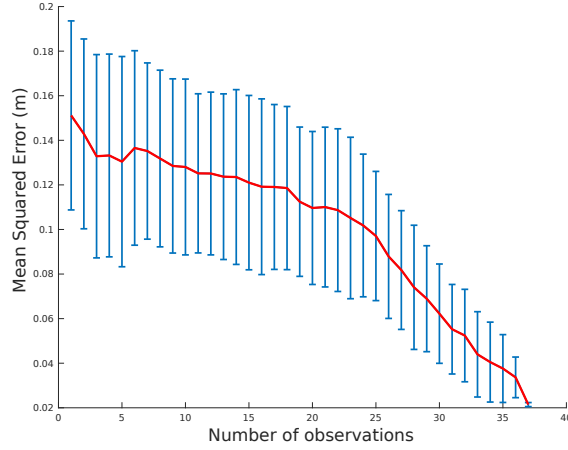


Figure 2.10: Mean squared prediction error (red curve) is reduced as more balls are observed. The ball observations are used until contact with racket occurs and the results are averaged over 100 different real ball trials. Correcting for ball prediction error is critical for a robust table tennis performance, as the balls typically come with a high spin. Balls seem to lose some spin after rebound and the prediction error decreases faster. In this case this phenomenon can be observed after about 25 ball observations, where the change in the average slope of the red curve can be seen.

maximum of 10 ball corrections till the ball passes the robot. The ball-launcher gives in addition a lot of *topspin* to the ball. This makes the corrections provided by the repeated optimization critical, as the ball models (2.9) – (2.14) are unable to capture some of the aerodynamic effects due to spin. Figure 2.10 shows the decrease in mean squared prediction error as more ball observations are acquired.

Discussion of Results

We compare and evaluate the performance of the two players FP and DP in the robot table tennis setup, see Figures 2.11 - 2.13. Results are averaged over 200 trials where the ball-launcher is fixed at different positions or is oscillating, and the robot is placed at three different initial postures. For FP, we also consider the variation in performance due to selecting different desired landing positions and landing times. Overall, FP is able to return about 40 – 60% of the balls to the opponent’s court. Setting the desired landing position on the right side of the table, with a desired landing time of $T_{\text{land}} = 0.4$ seconds, leads to the best performance ($\sim 60\%$) in our table tennis setup. The ball landing distribution follows roughly an isotropic Gaussian distribution with variance $\sigma_{\text{land}}^2 = 0.2m$ and a mean offset of $\|\mu_{\text{land}} - \mathbf{b}_{\text{goal}}\| = 0.25m$, see Figure 2.12b. Increasing the time and setting the desired landing position closer to the center of the opponent’s court makes the player less robust, decreasing the accuracy down to 40 – 50% and increasing the variance of the landing locations, see Figure 2.12a. We believe this decrease in the performance is due to inaccuracies in the racket model.

The *Defensive Player* (DP) is able to return about 80 – 90% of the balls, the performance varying depending on the incoming balls and the ballgun settings. The gain in accuracy is due to the increased flexibility of the algorithm, as well as the additional resting posture optimization

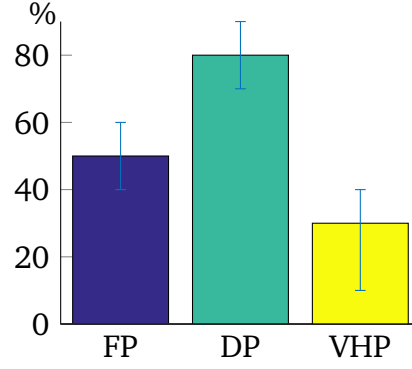


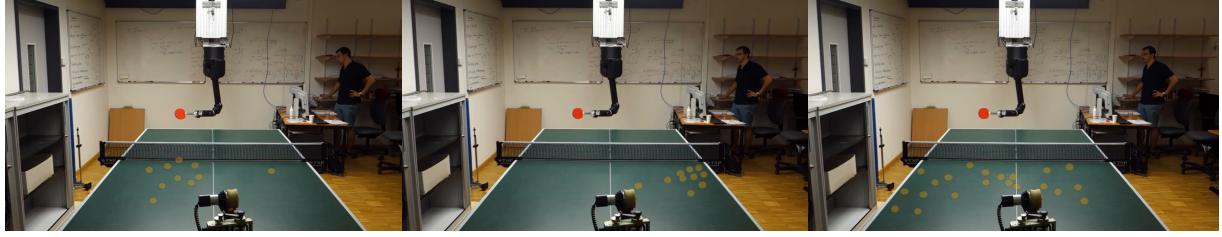
Figure 2.11: Summary of real robot table tennis experiment results comparing three table tennis players. Bar plot values show the successful return % averaged over different starting postures and initial ball positions. The error bars indicate the standard deviation over a total of 200 trial runs.

which simplifies the task significantly. The algorithm finds counterintuitive resting postures that lead to smaller movements with less control error, see Figure 2.14 for four consecutive trials of DP. The ball landing distribution in this case is roughly uniform but shifted to the left side of the table, with an offset mean of about $0.15m$, see Figure 2.12c. The duration of the returning trajectory $T_{\text{rest}} = 1.0s$ for all players. The weighting matrix \mathbf{R} is set to identity and the weights for hitting and landing penalties are both set to ten: $\alpha_{\text{hit}} = 10, \alpha_{\text{land}} = 10$.

VHP can return about 10 – 40% of the balls. The best setting for the hitting plane location depends strongly on the ballgun settings, which affect the distribution of the incoming ball. In our experiments the hitting plane at $y = 30$ cm in front of the robot lead to the best performance ($\sim 40\%$). However, the accuracy can drop down significantly (to 10%) if the ballgun is oscillating, or the initial ball velocities are not appropriate for the particular VHP setting. For all three algorithms, without applying any corrections, the robot is able to hit most balls but cannot return most balls successfully to the other side (only 5% of the balls are returned). Applying the corrections about three times, and at least once after rebound, increases the performance to the indicated values, see Figure 2.11. This indicates that the rebound model chosen might not be accurate with high topspins that the ballgun imparts to the ball (around 3000 rpm).

Two example trials are shown in Figure 2.13. The deviation from the desired hitting point, shown as an orange dot, was for the first example within three cm of the racket center, resulting in a hit. The deviations in the reference velocities are higher and lead to approximately 10 cm/s difference in Cartesian space. The successful strike and the landing on the opponent’s court can be seen in the upper right figure. In the first example, player FP tries to return the ball to the right side of the opponent’s court, with a desired landing time of $T_{\text{land}} = 0.4$ seconds. The blue dots are the ball observations acquired from cameras 3 and 4, which are located on the corners of the ceiling on the robot side. In the second example (screenshots 3 – 6), the player DP also returns the ball successfully, but unlike the other player, DP does not bring the robot back to the same initial posture. A video showing some example performances of the two players is available online: https://youtu.be/_kcTvD29M80. A longer version is also available in: <https://youtu.be/WlrmFaX705I>.

Control errors on the joint positions and velocities for this example are shown in Figure 2.15. After the desired trajectories are calculated, high gain PD-control is applied along



(a) FP with the desired ball landing location at the center of the opponent's court.

(b) FP with the desired ball landing location at the right side.

(c) DP with a more flexible returning criterion.

Figure 2.12: Overall, FP is able to return about 40 – 60% of the balls to the opponent's court. Setting the desired landing position on the right side of the table, with a desired landing time of $T_{\text{land}} = 0.4$ seconds, leads to the best performance ($\sim 60\%$) in our table tennis setup. Some example landing locations are indicated in orange in (b). Setting the desired landing position closer to the center of the opponent's court decreases the accuracy down to 40 – 50%, increasing also the variance of the landing locations, as shown in (a). DP in (c) with a landing accuracy of 80% has the highest variance in terms of the ball landing locations, as its returning criterion considers the whole opponent's court.

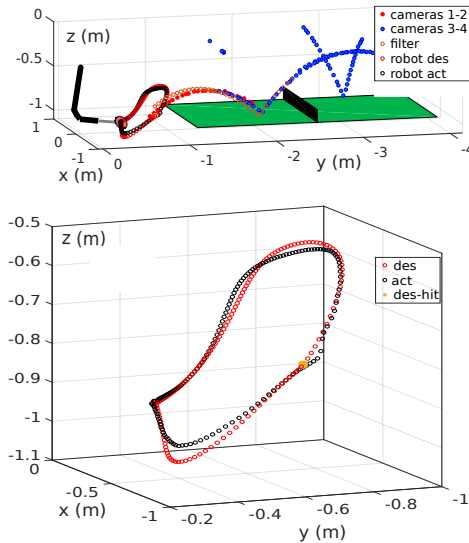
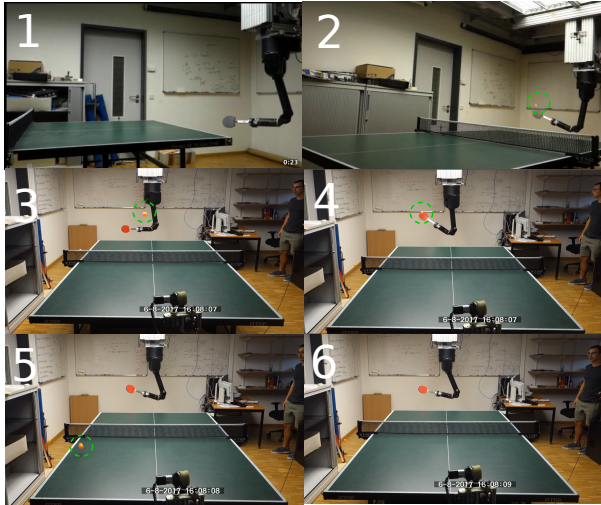


Figure 2.13: Two example table tennis trials recorded in the table tennis setup are shown on the left hand side. The top two screenshots show the *Focused Player* (FP) in action, and the bottom four the *Defensive Player* (DP). Unlike FP, DP does not bring the robot back to the same initial posture (screenshots 3 vs. 6). Successful strike and the valid landing on the opponent's court for DP can be seen in the screenshots 4 – 5. Balls are highlighted with green dashed circles for visibility. The plot in the upper right figure shows the recordings from the cameras and the robot sensors, corresponding to the hitting movement in screenshots 1 and 2. The blue dots are the ball observations coming from cameras 3 and 4. The desired Cartesian trajectory is drawn in red, and the actual trajectory, in black.

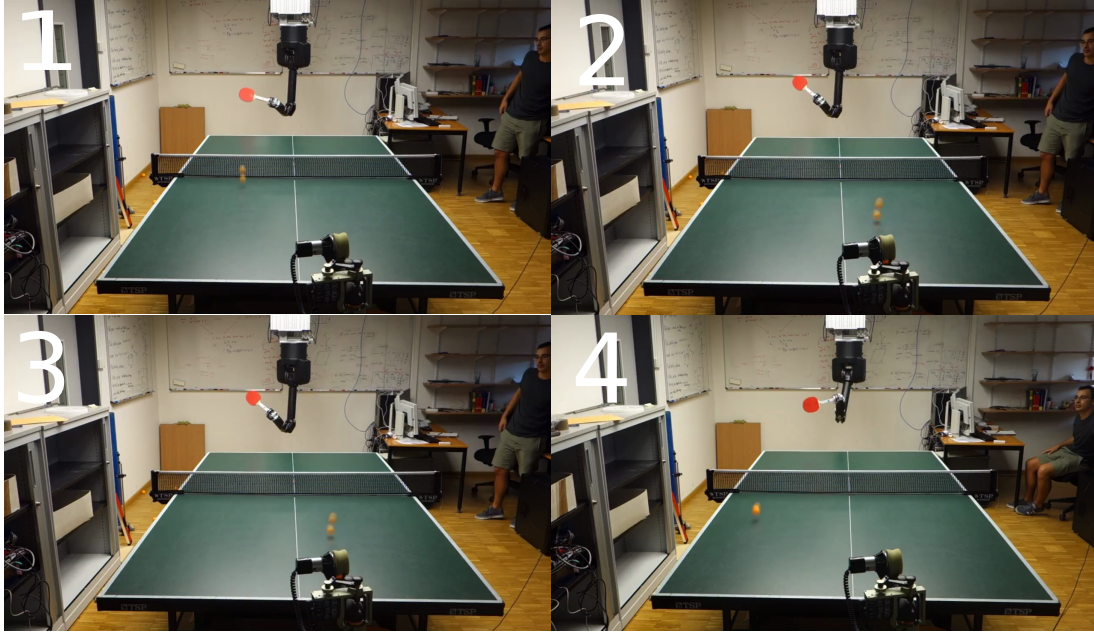


Figure 2.14: Four consecutive lands are shown for the *Defensive Player* (DP). In each trial, the arm goes back to a different resting posture.

with an inverse dynamics controller (computed-torque). The inverse dynamics model is not very precise, but the feedback with high gains compensates for it well, especially in the shoulders and the elbow.

2.7 Conclusion

In this chapter we have presented two new algorithms (FP and DP) for generating table tennis striking trajectories that extend previous work in table tennis strike movement generation.

2.7.1 Summary of the Contributions

The two table tennis players use an optimal-control based approach for generating striking trajectories. The striking and return trajectories are third order polynomials that intercept the ball at the optimized hitting point at the optimized hitting time. Unlike previous approaches, our optimization based framework respects the joint limits, while leading to efficient movements with low accelerations. Furthermore, by varying the hitting time T the problem of finding feasible joint trajectories is simplified. Further constraints can be easily imposed on the system, and we have considered, for instance, racket constraints for FP and an additional resting posture optimization for DP.

The optimizations can be run online in the robotic setup shown in Figure 4.1 and given new joint position and ball position measurements, the trajectories can be updated. Correcting for new ball positions, by repeating optimization, makes our table tennis players more robust to execution errors and inaccuracies in ball estimation & prediction. We show the performance of our two table tennis players in the real robot platform and compare with previous approaches.

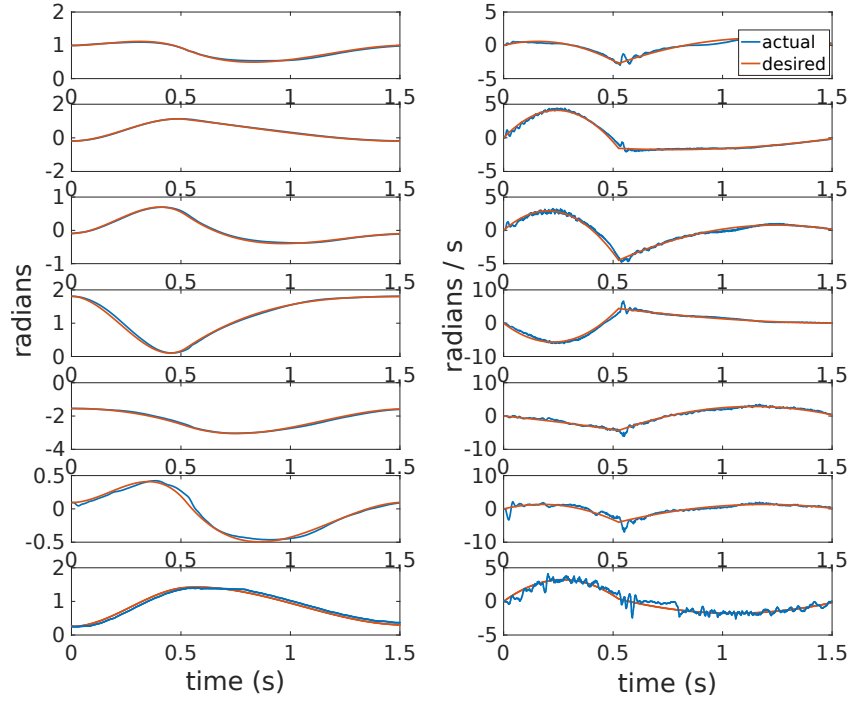


Figure 2.15: Tracking errors are shown for each joint. The desired joint positions and velocities are tracked with a PD controller. The deviation from the desired hitting point, shown as an orange dot in Figure 2.13, was for this example within three centimeters of the racket center, resulting in a hit.

2.7.2 Outlook & Future Work

The two players *Focused Player* and *Defensive Player* can generate trajectories more flexibly than before and lead to two different play-styles which could potentially be utilized by a higher-level strategy. We believe that this is a promising direction, where a higher level learning algorithm could switch between different trajectory generation schemes. The weights and the additional parameterization for the two algorithms can be explored based on feedback on the robot's performance. Reinforcement learning [42] with rewards based on observed ball landing positions, provides a suitable framework to tune the proposed algorithms' performance online.

Finally, the cost functionals that we have introduced consider the accelerations as the quantity to be minimized. Whenever the cancellation in feedback linearization is imperfect due to inaccurate robot dynamics models, execution errors will prevent the robot from achieving the desired trajectories or the minimal accelerations. A more robust and adaptive way to include execution errors in trajectory generation can be considered in future work.

3 Optimizing Execution of Robot Striking Movements with Learning Control

Most dynamic tasks in robotics include a *tracking* component, where the system is controlled to follow a desired reference trajectory. Robot table tennis [12],[2], in particular involves the generation of fast dynamic trajectories with high accelerations. These trajectories can be optimized well on the kinematics level, but reaching the target state and returning the ball requires accurate tracking of these hitting movements. Computing the appropriate control inputs for tracking can be a challenging task, especially when using cable-driven arms, such as the Barrett WAM shown in Figure 4.1, due to mechanical compliance and low bandwidth.

3.1 Introduction

Iterative Learning Control (ILC) is a control theoretic learning framework restricted to tracking (time-varying) reference trajectories[43]. In ILC, the goal is to improve the tracking performance, reducing the future deviations along the fixed trajectory, and ultimately driving them to the minimum possible. After observing the deviations from the reference trajectory at each iteration, the errors are fed back to the (feedforward) control inputs for the next iteration. Any available dynamics models can be incorporated easily during these updates, see e.g., [44], [45]. ILC has been used successfully in several robotics tasks to improve trajectory tracking performance under unknown repeating disturbances and model mismatch [43].

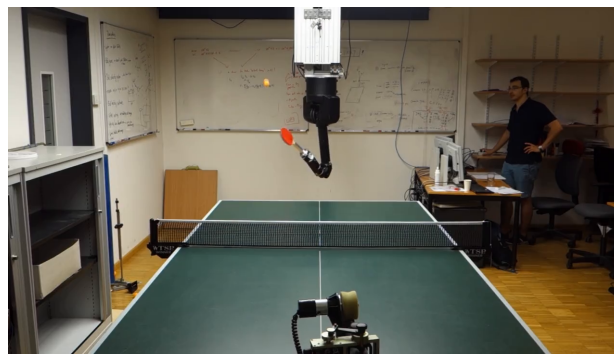


Figure 3.1: Our robot table tennis platform where a seven degree of freedom Barrett WAM arm is shown facing a ball-launcher. The ball is tracked using four cameras on the ceiling. Whenever a ball is approaching the robot, reference trajectories are computed on-line in order to return the ball to a desired location on the opponent's court. Such trajectories can be optimized on the kinematics level [2], however it is hard to execute them accurately without having access to accurate dynamics models. Iterative Learning Control, using inaccurate models, can still lead to an efficient approach for learning to track these trajectories.

While there have been many impressive applications of reinforcement learning (RL) [42] to learn robotic tasks [46], RL remains to be computationally and information-theoretically hard in general. Much of control, on the other hand, can be reduced to supervised learning, with the appropriate reference trajectories. By making good use of existing, albeit imperfect, models and smooth reference trajectories with ILC, learning efficiency in robotics tasks can be improved significantly. However, it is rather difficult to ensure a stable learning performance in practice, see Figure 3.2 for an illustration.

In this chapter, we introduce a new model-based learning approach for tracking a variety of fast, dynamic movements stably, while maintaining learning and computational efficiency. Stability of the updates, or the probability of update monotonicity, is increased by making use of dynamics model covariance estimates. We refer to this as *caution* throughout the text, and the resulting algorithm is cautious precisely in this sense. A cautious learning control algorithm can hence be defined as one that incorporates a probabilistic notion of stability (in the iteration domain) during decision making, for the control input updates. This property proves to be critical, as we show the learning performance for fast robot table tennis striking movements. The proposed Bayesian approach, using the posterior over the dynamics model parameters, maintains both adaptation and caution in model-based ILC, while being efficient in terms of learning performance and computational complexity.

Our contributions can be stated succinctly as follows: we propose a new adaptive and cautious model-based ILC algorithm, that is implemented efficiently using a recursive formulation. More specifically, the existing model-based recursive ILC approach of Amann et al. [45], introduced briefly in Section 3.2, is extended to include adaptation (by using Linear Bayes Regression on the errors) and caution (or in other terms, robustness to modelling errors, which shows itself as learning stability in the iteration domain). The proposed approach minimizes an expected quadratic cost term over the trajectory deviations, which still yields a closed-form solution, resulting in a cautious yet efficient learning performance. In the closed-form solution, the covariances of the learned local linear models are employed as adaptive regularizers. The expected cost minimization distinguishes the framework from more conservative min-max approaches, such as the robustly convergent ILC proposed in the literature (using H_∞ and μ -synthesis techniques [47]). Related work in the theory and practice of ILC, as well as some more general applications of learning in robotics tasks, are briefly mentioned in the next subsection. Before introducing the expected cost minimization framework in Section 3.4, we discuss model adaptation in Section 3.3 with linear time-varying models and show that Broyden’s method [3] can be derived from Linear Bayesian Regression (LBR) as the forgetting factor goes to zero. Thus, the proposed approach belongs to the family of Quasi-Newton ILC methods [48].

The resulting adaptive and cautious ILC algorithm, called *bayesILC* is described in Section 3.5, and extensions are discussed for additional robustness to nonrepetitive disturbances. Derivations for the recursive and cautious learning control update are left to the Appendix A.4. We evaluate *bayesILC* first in extensive simulations in Section 3.6, showing that the proposed method is stable, efficient and can outperform other state-of-the-art learning approaches. We then present online learning results on our robot table tennis platform for tracking dynamic hitting movements. We discuss the real robot learning results in Section 3.7 and conclude with brief mentions of promising future research directions.

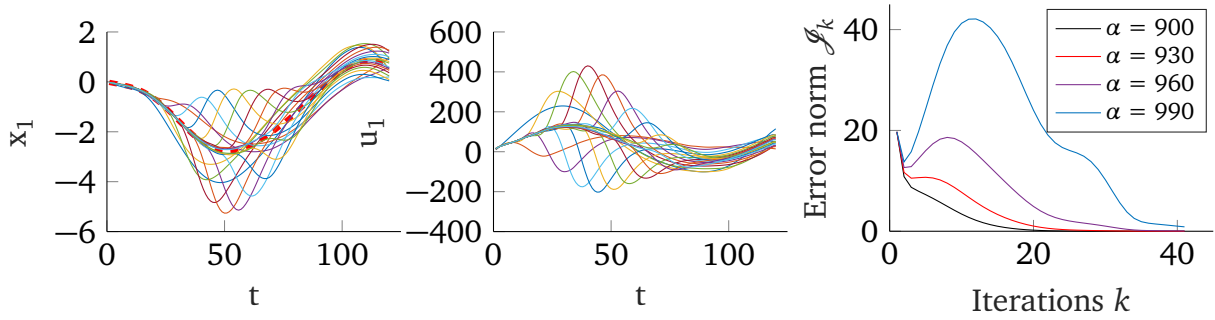


Figure 3.2: Learning performance of ILC, using inaccurate models without incorporating a notion of uncertainty, may not be monotonic in practice. One can observe ripples that move through the trajectory which can cause instability or damage the robot. In simulations we can create this effect easily by increasing the spectral norm of the difference between the nominal and the actual (lifted) dynamics matrices. The desired trajectory for the first state x_1 is shown in dashed red on the left-hand side for a two dimensional linear time invariant system. The second plot shows the ILC feed-forward commands for this particular trajectory and state. The third plot shows the Frobenius norm of the trajectory deviations, J_k , plotted over the iterations k . The nonmonotonicity of the learning performance is aggravated, as the mismatch scale α controlling the spectral norm of the difference is increased. Increasing α further can prevent even asymptotic stability. The curves were generated by direct inversion of the (lifted) model. Our proposed Bayesian approach, on the other hand, minimizing the expected cost throughout the iterations, uses the posterior over the dynamics model parameters to make more cautious decisions.

3.1.1 Related Work

Since the eighties, there have been many different Iterative Learning Control update laws proposed, with the D-type update law of Arimoto et al. [49] being one of the first. See [43] and [44] for reviews and categorization of the different update laws. Theoretically, most ILC updates are *linear repetitive processes* that can be analyzed using 2D-systems analysis [50], i.e., assuming the desired trajectory is fixed and the initial conditions can be reset perfectly, the error over the iterations has a (discrete) dynamics of its own. Stability of the ILC updates and monotonic convergence in particular can then be studied using dynamical systems theory. These notions also play an important role in the design of practical ILC algorithms. See [43], [51] for a discussion and [52] for insight into convergence and stability issues appearing in an implementation.

Stability issues and the induced oscillations (see Figure 3.2 for a simple simulation example) can easily damage the system to be controlled. For instance, joint limits can be exceeded in a robotics application or other task-imposed state constraints can be violated. Such issues complicate the application of ILC in high dimensional robotics problems. In practice, additional complications can occur, such as varying initial conditions, violating the assumptions made in most of the ILC literature. Robustness to varying initial conditions were considered e.g., in [53], [54], [55]. For additional robustness to nonrepeating disturbances or noise, a robust feedback controller should be used alongside ILC, see e.g., [56], [52].

Methods that learn to track (periodic or episodic) trajectories need to compensate for modeling uncertainties and other repetitive disturbances acting on the system to be controlled.

However, methods that can efficiently learn the dynamics are model-based (e.g. most of optimization-based ILC [45],[43]) and at least require knowing the correct signs for the linearized dynamics of the system [57], [58].

When executing model-based learning algorithms on dynamical systems, it is essential for stability and safety to incorporate a notion of model uncertainty. Otherwise the learning algorithms can be overconfident and quickly go unstable [52]. One way to achieve a more stable performance in ILC is to filter the high-frequency updates. These robust methods are mostly known as Q-filtering [43] and typically incur a trade-off between stability and performance: the system will often fail to converge to the minimal steady-state error. In this chapter, we use a different (probabilistic) approach to increase the stability margins of model-based ILC that does not incur such a trade-off. To that end, we expand on the previous work of Amann et al. [45], one of the first model-based ILC approaches introducing an optimal-control based ILC design. The recursive implementation first introduced in this chapter closely relates to numerically-stable plant-inversion approaches [59]. We extend the recursive formulation to include adaptation and caution: adaptation of the model parameter means and variances are performed at each iteration using Linear Bayes Regression. The resulting Bayesian approach, minimizing the expected cost throughout the iterations, uses the posterior over the dynamics model parameters to make more cautious decisions.

Model adaptation in ILC can be studied in the context of solving nonlinear equations. Tracking a fixed reference perfectly corresponds to solving for the control inputs that drive the deviations to zero. Hence, Quasi-Newton methods such as the Broyden's method [3] and generalized secant method [60] were proposed as adaptation methods in the ILC literature to update the plant dynamics. Broyden's method, without having access to the gradients of a black-box function $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, maintains a Jacobian matrix approximation \mathbf{F} . The matrix \mathbf{F} is updated at each iteration k in order to satisfy the *secant equation*

$$\mathbf{f}_k - \mathbf{f}_{k-1} = \mathbf{F}_k(\mathbf{x}_k - \mathbf{x}_{k-1}), \quad (3.1)$$

which can then be inverted to yield¹

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{F}_k^\dagger \mathbf{f}_k. \quad (3.2)$$

Convergence under restrictive assumptions have been shown for Broyden's method. For solving systems of nonlinear equations, arguably efficiency rather than stability or monotonic convergence is of importance, and a simple trust-region approach (based on a merit function) suffices to improve stability. We will show how Broyden's method can be seen as a limiting case of Linear Bayesian Regression in Section 3.3. The proposed method thus belongs to the family of Quasi-Newton optimization methods [3], where the black-box nature of the Quasi-Newton approaches is augmented to include caution during the ILC updates: monotonic convergence, or update stability in the iteration domain, is of paramount importance in robotics tasks.

An application of model-based ILC to reject repeating disturbances was shown in quadcopter flight [61], where a constrained convex optimization with imposed control input limits was solved, rather than a direct inversion of the nominal model dynamics. An impressive application of ILC to a robotic surgical task was presented in [62] utilizing an EM-based ILC update law. ILC was also combined with robust observers to control a heavy-duty hydraulic arm in an excavation task [63].

¹ Broyden's method can also directly update the inverse.

Besides ILC, another learning framework that learns inaccurate models for control is model-based Reinforcement Learning. Including variance fully in the decision-making process can result in efficient and stable learning [64]. However such involved procedures exhibit computational runtime difficulties and have not been implemented in high-dimensional real-time robotics tasks.

3.2 Problem Statement and Background

Most tasks in robotics can be learned more efficiently whenever feasible trajectories are available. Learning-based control approaches can then focus on tracking these trajectories without relying on accurate models. The goal in trajectory tracking is to track a given reference $\mathbf{r}(t)$, $0 \leq t \leq T$, by applying the control inputs $\mathbf{u}(t)$. In dynamic robotic tasks, the references are often in the combined state space of joint positions and velocities $(\mathbf{q}^T, \dot{\mathbf{q}}^T)^T \in \mathcal{Q} \subset \mathbb{R}^{2n}$, and the control inputs $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$ are applied for each joint of the robot, i.e., $m = n$. The reference trajectories in table tennis, for instance, enable the execution of hitting and striking motions, e.g., forehand and backhand strikes. Such trajectories can be generated online with nonlinear constrained optimization [2]. Finding the right control inputs to track them accurately is the focus of Iterative Learning Control (ILC).

Linearizing an Inaccurate Model

Consider a nonlinear robot dynamics model

$$\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}), \quad (3.3)$$

e.g., for rigid body dynamics of the form

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\{\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})\}, \quad (3.4)$$

where on the right-hand side are the inverse of the inertia matrix $\mathbf{M}(\mathbf{q})$, the Coriolis and centrifugal forces $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$, and the vector of gravitational forces $\mathbf{G}(\mathbf{q})$. This nonlinear dynamics model can be linearized around a given joint space trajectory $\mathbf{r}(t)$, $0 \leq t \leq T$ with nominal inputs $\mathbf{u}_{\text{IDM}}(t)$ calculated using the inverse dynamics model [36]. We then obtain the following linear time-varying (LTV) representation

$$\dot{\mathbf{e}}(t) = \mathbf{A}(t)\mathbf{e}(t) + \mathbf{B}(t)\delta\mathbf{u}(t) + \mathbf{d}(t, \mathbf{u}), \quad (3.5)$$

where the state vector is the joint angles and velocities $\mathbf{x} = [\mathbf{q}^T, \dot{\mathbf{q}}^T]^T$, the state error is denoted as $\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{r}(t)$, the deviations from the nominal inputs are $\delta\mathbf{u}(t) = \mathbf{u}(t) - \mathbf{u}_{\text{IDM}}(t)$ and the continuous time-varying matrices are

$$\mathbf{A}(t) = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right|_{(\mathbf{r}(t), \mathbf{u}_{\text{IDM}}(t))}, \mathbf{B}(t) = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \right|_{(\mathbf{r}(t), \mathbf{u}_{\text{IDM}}(t))}. \quad (3.6)$$

In the error dynamics (3.5), the additional (unknown) term $\mathbf{d}(t, \mathbf{u})$ accounts for the disturbances and the effects of the linearization (i.e., higher order terms). We can discretize (3.5-3.6) with step size δ , $N = T/\delta$ and step index $j = 1, \dots, N$ to get the following discrete-time linear system

$$\mathbf{e}_{j+1} = \mathbf{A}_j \mathbf{e}_j + \mathbf{B}_j \delta \mathbf{u}_j + \mathbf{d}_{j+1}, \quad (3.7)$$

where the matrices $\mathbf{A}_j, \mathbf{B}_j$ are the discretizations of (3.6). Conventional (discrete) ILC algorithms learn to compensate for the errors incurred along the trajectory by updating the control inputs $\delta \mathbf{u}_j$ iteratively.

Whenever we refer to the outcome of a particular iteration k , we will use the first subindex for iterations and the second subindex will be used to denote the (discrete) time step, i.e., the vectors $\mathbf{e}_{k,j} \in \mathbb{R}^{2n}$, $\delta \mathbf{u}_{k,j} \in \mathbb{R}^m$ denote the deviations and control input compensations at the time step j during iteration k , respectively. The control commands applied at iteration $k+1$ as

$$\mathbf{u}_{k+1,j} = \mathbf{u}_{k,j} + \delta \mathbf{u}_{k,j}, \quad (3.8)$$

are computed using the deviations $\mathbf{e}_{k,j}$ at iteration k .

Recursive Norm-Optimal ILC

Norm-optimal ILC uses the discrete LTV model in (3.7) to minimize the next iteration errors, where the computed control inputs are optimal with respect to some vector norm. These approaches based on optimality criteria can learn efficiently by taking advantage of the inaccurate models. Batch methods that compute the next iteration compensations stack the model matrices together to compute (a possibly weighted and dampened) pseudoinverse of this block lower-diagonal matrix. As an alternative, some methods use convex programming to compute these optimal compensations under additional constraints.

The condition of this *lifted* model matrix typically grows exponentially with the horizon size N and computing the pseudoinverse stably becomes very difficult. Downsampling trajectories restores the condition number and a stable inversion becomes much more manageable, at the cost of reduced tracking performance. As a better alternative, optimization-based approaches, depending on the particular optimizer, may avoid computing the pseudoinverse. However such approaches can still be computationally intensive, and may not be suitable for online learning.

As an alternative, the authors in [45] have shown that the direct batch inversion of the lifted model matrix can be avoided by recursively computing the ILC compensations (in one pass) using the Linear Quadratic Regulator (LQR) for disturbance rejection [65]. After estimating the disturbances \mathbf{d}_{j+1} at the k 'th trial, the optimal control problem for tracking a desired trajectory can be written as

$$\begin{aligned} \min_{\delta \mathbf{u}} \quad & \sum_{j=1}^N \mathbf{e}_{k+1,j}^T \mathbf{Q}_j \mathbf{e}_{k+1,j} + \delta \mathbf{u}_{k,j}^T \mathbf{R}_j \delta \mathbf{u}_{k,j}, \\ \text{s.t.} \quad & \mathbf{e}_{k+1,j+1} = \mathbf{A}_j \mathbf{e}_{k+1,j} + \mathbf{B}_j \mathbf{u}_{k+1,j} + \mathbf{d}_{j+1}. \end{aligned} \quad (3.9)$$

Reduction of the ILC problem to the known LQR solution has not attracted much attention however from the control and learning communities, since it was not clear how to study stability and convergence in this formulation.

3.3 Model Adaptation

Whenever there is model-mismatch, the (linearized) models cannot be assumed to hold accurately around the reference trajectory. There is hence a risk that the learning process described in the previous subsections will not be stable. As a remedy, in this section we propose a natural Bayesian adaptation of model matrices with Linear Bayesian Regression (LBR) and discuss different alternatives in the context of robotics.

3.3.1 Recursive Estimation of Model Matrices

The observed deviations from the trajectory at iteration k , $\mathbf{e}_{k,j}$, can be used to update the discrete-time LTV model matrices $\mathbf{A}_{k,j}, \mathbf{B}_{k,j}$ that describe the nonlinear dynamics around the trajectory, to first order. Instead of estimating all the parameters together in a costly estimation procedure, the model matrices $\mathbf{A}_{k,j}, \mathbf{B}_{k,j}$ can rather be updated separately for each $j = 1, \dots, N$, given the smoothened errors $\hat{\mathbf{e}}_{k,j}$

$$\hat{\mathbf{e}}_{k,j+1} = \mathbf{A}_{k,j} \hat{\mathbf{e}}_{k,j} + \mathbf{B}_{k,j} \mathbf{u}_{k,j} + \mathbf{d}_{j+1}, \quad (3.10)$$

which can be rewritten using the Kronecker product and the vectorization operator as follows

$$\begin{aligned} \hat{\mathbf{e}}_{k,j+1} - \hat{\mathbf{e}}_{k-1,j+1} &\approx \mathbf{X}_{k,j} \text{vec}[\mathbf{A}_{k,j}, \mathbf{B}_{k,j}], \\ \mathbf{X}_{k,j} &= \text{vec}[\hat{\mathbf{e}}_{k,j} - \hat{\mathbf{e}}_{k-1,j}, \delta \mathbf{u}_{k,j}]^T \otimes \mathbf{I}. \end{aligned} \quad (3.11)$$

If we incorporate the belief (including the uncertainty) about the linear dynamics models as Gaussian priors in LBR

$$\begin{aligned} \boldsymbol{\theta}_{k,j} &= \text{vec}[\mathbf{A}_{k,j}, \mathbf{B}_{k,j}], \\ \mathbf{y}_{k,j} &= \hat{\mathbf{e}}_{k,j+1} - \hat{\mathbf{e}}_{k-1,j+1}, \\ \rho(\boldsymbol{\theta}_{k,j} | \mathbf{y}_{k,j}) &\propto \rho(\mathbf{y}_{k,j} | \boldsymbol{\theta}_{k,j}) \rho(\boldsymbol{\theta}_{k,j}), \\ \rho(\boldsymbol{\theta}_{k,j}) &= \mathcal{N}(\boldsymbol{\theta}_{k,j} | \boldsymbol{\mu}_{k,j}, \boldsymbol{\Sigma}_{k,j}), \end{aligned} \quad (3.12)$$

with a Gaussian likelihood function

$$\rho(\mathbf{y}_{k,j} | \boldsymbol{\theta}_{k,j}) = \mathcal{N}(\mathbf{y}_{k,j} | \mathbf{X}_{k,j} \boldsymbol{\theta}_{k,j}, \sigma^2 \mathbf{I}), \quad (3.13)$$

the models parameter means $\boldsymbol{\mu}_{k,j}$ and variances $\boldsymbol{\Sigma}_{k,j}$ can be updated as

$$\begin{aligned} \boldsymbol{\Sigma}_{k,j} &= \left(\frac{1}{\sigma^2} \mathbf{X}_{k,j}^T \mathbf{X}_{k,j} + \boldsymbol{\Sigma}_{k-1,j}^{-1} \right)^{-1}, \\ \boldsymbol{\mu}_{k,j} &= \boldsymbol{\Sigma}_{k,j} \left(\boldsymbol{\Sigma}_{k-1,j}^{-1} \boldsymbol{\mu}_{k-1,j} + \frac{1}{\sigma^2} \mathbf{X}_{k,j}^T \mathbf{y}_{k,j} \right). \end{aligned} \quad (3.14)$$

Smoothened position and velocity error estimates can be obtained, for example, using a zero-phase Butterworth filter.

Relation to Broyden's method

Broyden's method [3] can be seen as a limiting case of LBR. The mean estimates in (3.14) are also the solutions of the following linear ridge regression problem

$$\min_{\boldsymbol{\theta}} \frac{1}{\sigma^2} \|\mathbf{y}_{k,j} - \mathbf{X}_{k,j} \boldsymbol{\theta}\|_2^2 + (\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j}) \boldsymbol{\Sigma}_{k,j}^{-1} (\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j}), \quad (3.15)$$

and as $\sigma^2 \rightarrow 0$ we get the (weighted) Broyden's update for one iteration, which, written in vectorized form, is solving independently for every time step

$$\min_{\boldsymbol{\theta}} (\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j}) \boldsymbol{\Sigma}_{k,j}^{-1} (\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j}), \quad (3.16)$$

$$\text{s.t. } \mathbf{y}_{k,j} = \mathbf{X}_{k,j} \boldsymbol{\theta}. \quad (3.17)$$

Broyden's method is too sensitive to the sensor noise in robotics tasks as it satisfies the secant rule (3.17) exactly. On the other hand, LBR in (3.14) for fixed noise parameter σ^2 , is using *all* of the past iteration data equally. The norm of the covariance decreases monotonically in each update. For unknown dynamic systems that are highly nonlinear but smooth, to prevent premature shrinking of the covariance matrix, a better alternative is to set an *exponential weighting* in the adaptation. For a fixed forgetting factor $\lambda \in [0, 1]$, the update in (3.14) becomes

$$\begin{aligned} \boldsymbol{\Sigma}_{k,j} &= \left(\frac{1}{\sigma^2} \mathbf{X}_{k,j}^T \mathbf{X}_{k,j} + \lambda \boldsymbol{\Sigma}_{k-1,j}^{-1} \right)^{-1}, \\ \boldsymbol{\mu}_{k,j} &= \lambda \boldsymbol{\Sigma}_{k,j} \boldsymbol{\Sigma}_{k-1,j}^{-1} \boldsymbol{\mu}_{k-1,j} + \frac{1}{\sigma^2} \boldsymbol{\Sigma}_{k,j} \mathbf{X}_{k,j}^T \mathbf{y}_{k,j}. \end{aligned} \quad (3.18)$$

The forgetting factor λ is used to perform exponential weighting of the previous iteration data. As $\lambda \rightarrow 0$, we get the (unweighted) Broyden's method², and as $\lambda \rightarrow 1$, (3.18) reduces to (3.14). Hence, our proposed adaptation law (3.18) can be embedded within a one-parameter family of Quasi-Newton ILC methods, where the forgetting factor parameter trades-off adaptation flexibility and robustness to noise. At the one end of the spectrum, Broyden's method adapts flexibly and aggressively to the latest data at the cost of being very sensitive to noise. This can be alleviated with a judicious choice of the forgetting factor. See Figure 3.3 for an illustration.

3.3.2 Imposing structure

The structure in the forward dynamics model (3.4) is not considered in the update rule (3.18): any change in the control inputs in this model directly affects the instantaneous joint accelerations, and only indirectly the joint velocities in the future time steps. By differentiating the smoothened joint velocities, one can instead impose the following regression model

$$\ddot{\mathbf{q}}_{k,j} - \ddot{\mathbf{q}}_{k-1,j} \approx \mathbf{A}_k(\delta j) \mathbf{e}_{k,j} + \mathbf{B}_k(\delta j) \delta \mathbf{u}_{k,j}, \quad (3.19)$$

where we dropped the hat for notational simplicity. The *continuous* model matrices $\mathbf{A}_k(\delta j), \mathbf{B}_k(\delta j)$ are members of a reduced parameter space, i.e., $\mathbf{A}_k(\delta j) \in \mathbb{R}^{n \times 2n}, \mathbf{B}_k(\delta j) \in$

² Unlike the case where $\sigma^2 \rightarrow 0$, this equivalence is valid for all the subsequent iterations as well. It can be seen more easily from the filter form of (3.18).

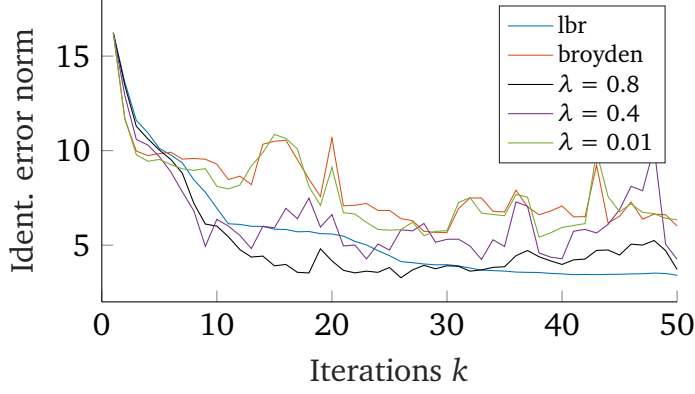


Figure 3.3: Broyden’s method [3], which can be considered as an adaptation framework within ILC, is a limiting case of Linear Bayesian Regression (LBR). As the forgetting factor λ of an exponentially weighted LBR model goes to zero, LBR transitions to Broyden’s method. Broyden’s method is very sensitive to noise and adapts very aggressively. Throughout the chapter, we discuss and evaluate several adaptation laws, that are less sensitive to noise but are still flexible. The Figure shows the evolution of the identification error norm for an unknown linear time-varying system. The Frobenius norm of the difference between the adapted model matrices ($\mathbf{A}_{k,j}$ and $\mathbf{B}_{k,j}$) and the actual (fixed) matrices (denoted as identification error norm) are plotted for each iteration $k = 1, \dots, 50$.

$\mathbb{R}^{n \times m}$, $j = 1, \dots, N$. After regressing on the continuous model matrices as in (3.14), they can be discretized (as discussed before) to form the discrete-time model parameter means $\mathbf{A}_{k,j} \in \mathbb{R}^{2n \times 2n}$, $\mathbf{B}_{k,j} \in \mathbb{R}^{2n \times m}$ and covariances $\Sigma_{k,j}$.

As an alternative, note that the rigid body dynamics (3.3) is parameterized by the link masses, three link center of mass values and six inertia parameters. A total of ten parameters are used for each link to fully parameterize the inverse dynamics model

$$\mathbf{u} = \mathbf{M}(\mathbf{q}; \boldsymbol{\theta}) \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}; \boldsymbol{\theta}) \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}; \boldsymbol{\theta}), \quad (3.20)$$

which can be stacked for each $j = 1, \dots, N$ to form the regression model

$$\begin{aligned} \mathbf{U}_k &\approx \mathbf{Y}(\mathbf{Q}_k^{(0)}, \mathbf{Q}_k^{(1)}, \mathbf{Q}_k^{(2)}) \boldsymbol{\theta}_k, \\ \mathbf{U}_k &= \left(\mathbf{u}_{k,1}^T, \mathbf{u}_{k,2}^T, \dots, \mathbf{u}_{k,N}^T \right)^T, \\ \mathbf{Q}_k^{(l)} &= \left(\mathbf{q}_{k,1}^{(l)T}, \mathbf{q}_{k,2}^{(l)T}, \dots, \mathbf{q}_{k,N}^{(l)T} \right)^T, \quad l = 0, 1, 2, \end{aligned} \quad (3.21)$$

where $\boldsymbol{\theta}_k \in \mathbb{R}^{10n}$ appears *linearly*. The index l denotes the degree of the derivatives of the smoothed joint angles, i.e., $l = 0, 1, 2$ are used to denote the joint position, velocity and acceleration estimates in (3.21), respectively. Based on these joint estimates, only the link parameters are updated with LBR as in (3.14). The forward dynamics model³ (3.3) can then be used to sample the means and variances of the continuous LTV matrices, e.g., using Monte Carlo sampling. Discretization as discussed above converts the continuous-time model parameter

³ The forward dynamics model (3.3), unlike the inverse dynamics (3.21), depends *nonlinearly* on the link parameters.

means and variances into their discrete-time form. An advantage of this approach is to *compress* learning to a lower dimensional space, reducing the variance of the updates at the cost of an introduced bias. Moreover, since the link parameters are invariant throughout the iterations, such an update avoids the flexible yet independent adaptation of the model matrices for each j , and the necessity of introducing a forgetting factor.

3.4 Cautious Learning Control

The posterior model covariances $\Sigma_{k,j}$ can be used to make more *cautious* decisions within a stochastic control framework. The uncertainty of the model parameters can be seen as a *multiplicative* noise model and the ILC optimality criterion (3.9) can be extended to include expectations over them. The multiplicative noise model, unlike the additive noise case, does not lead to *certainty-equivalence*: the covariance estimates are incorporated in the decision rule. To see how the expected cost minimization leads to caution, note that

$$\mathbb{P}(\mathbf{e}_{k+1,j}^T \mathbf{Q}_j \mathbf{e}_{k+1,j} \geq \hat{\mathbf{e}}_{k,j}^T \mathbf{Q}_j \hat{\mathbf{e}}_{k,j}) \leq \frac{\mathbb{E}[\mathbf{e}_{k+1,j}^T \mathbf{Q}_j \mathbf{e}_{k+1,j}]}{\hat{\mathbf{e}}_{k,j}^T \mathbf{Q}_j \hat{\mathbf{e}}_{k,j}}, \quad (3.22)$$

which follows from Markov's inequality. Minimizing the upper bound forces the probability of nonmonotonicity to be low as well.

Expected Cost Minimization

For the expected cost case, where the expectation is taken over the *random variables* $\mathbf{A}_{k,j}$ and $\mathbf{B}_{k,j}$, for each j , the optimality criterion

$$\begin{aligned} \min_{\delta \mathbf{u}} \sum_{j=1}^N \mathbb{E}_{\mathbf{A}_{k,j}, \mathbf{B}_{k,j}} [\mathbf{e}_{k+1,j}^T \mathbf{Q}_j \mathbf{e}_{k+1,j} + \delta \mathbf{u}_{k,j}^T \mathbf{R}_j \delta \mathbf{u}_{k,j}], \\ \text{s.t. } \mathbf{e}_{k+1,j+1} = \mathbf{A}_{k,j} \mathbf{e}_{k+1,j} + \mathbf{B}_{k,j} \mathbf{u}_{k+1,j} + \mathbf{d}_{j+1}, \end{aligned} \quad (3.23)$$

can be solved recursively using dynamic programming [14]

$$\begin{aligned} \delta \mathbf{u}_{k,j} &= \mathbf{K}_{k,j} \mathbf{e}_{k+1,j} - \Phi_{k,j}^{-1} \ell_{k,j}, \\ \mathbf{K}_{k,j} &= -\Phi_{k,j}^{-1} \Psi_{k,j}, \\ \Phi_{k,j} &= \mathbb{E}_{\mathbf{B}_{k,j}} [\mathbf{B}_{k,j}^T \mathbf{P}_{k,j+1} \mathbf{B}_{k,j}] + \mathbf{R}_j, \\ \Psi_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j}, \mathbf{B}_{k,j}} [\mathbf{B}_{k,j}^T \mathbf{P}_{k,j+1} \mathbf{A}_{k,j}], \\ \ell_{k,j} &= \mathbb{E}_{\mathbf{B}_{k,j}} [\mathbf{B}_{k,j}^T \mathbf{P}_{k,j+1} (\mathbf{B}_{k,j} \mathbf{u}_{k,j} + \mathbf{d}_{j+1}) + \mathbf{B}_{k,j}^T \mathbf{b}_{k,j+1}], \end{aligned} \quad (3.24)$$

where $\mathbf{b}_{k,j}$ and the Ricatti matrices $\mathbf{P}_{k,j}$ evolve backwards according to

$$\begin{aligned} \mathbf{P}_{k,j} &= \mathbf{Q}_j + \mathbf{M}_{k,j} - \Psi_{k,j}^T \Phi_{k,j}^{-1} \Psi_{k,j}, \\ \mathbf{M}_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j}} [\mathbf{A}_{k,j}^T \mathbf{P}_{k,j+1} \mathbf{A}_{k,j}], \\ \mathbf{b}_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j}, \mathbf{B}_{k,j}} [\bar{\mathbf{A}}_{k,j}^T (\mathbf{b}_{k,j+1} + \mathbf{P}_{k,j+1} (\mathbf{B}_{k,j} \mathbf{u}_{k,j} + \mathbf{d}_{j+1}))], \end{aligned} \quad (3.25)$$

starting from $\mathbf{P}_{k,N} = \mathbf{Q}_N$ and $\mathbf{b}_{k,N} = \mathbf{0}$. The random closed loop system dynamics is given by the matrices

$$\bar{\mathbf{A}}_{k,j} = \mathbf{A}_{k,j} + \mathbf{B}_{k,j}\mathbf{K}_{k,j}. \quad (3.26)$$

By a direct comparison to the LQR solution to (3.9), it can be seen that the control input compensations $\delta \mathbf{u}_{k,j}$ in (3.24) are computed similarly, with the appropriate expectations added. The ILC update is decomposed into two components: a current-iteration feedback term $\mathbf{u}_{fb} = \mathbf{K}_{k,j}\mathbf{e}_{k+1,j}$ calculated using the iteration dependent Riccati equations and a feedforward, purely predictive term $\mathbf{u}_{ff} = -\Phi_{k,j}^{-1}\ell_{k,j}$, solved backwards for each $j = 1, \dots, N$. The feedforward terms are responsible for compensating for the estimated *random* disturbances \mathbf{d}_j , calculated using (3.10).

Cautious update (3.24) can be implemented without explicitly calculating the disturbances. If the disturbances are taken as random variables defined via the filtered errors $\hat{\mathbf{e}}_{k,j}$ of the last iteration

$$\mathbf{d}_{j+1} = \hat{\mathbf{e}}_{k,j+1} - \mathbf{A}_{k,j}\hat{\mathbf{e}}_{k,j} - \mathbf{B}_{k,j}\mathbf{u}_{k,j}, \quad (3.27)$$

the recursion can be simplified by introducing

$$\boldsymbol{\nu}_{k,j} = \mathbf{b}_{k,j} + \mathbf{P}_{k,j}\mathbf{e}_{k,j}. \quad (3.28)$$

The feedforward and feedback compensations $\delta \mathbf{u}_{k,j}$ can then directly be computed as

$$\begin{aligned} \delta \mathbf{u}_{k,j} &= \mathbf{K}_{k,j}(\mathbf{e}_{k+1,j} - \mathbf{e}_{k,j}) - \Phi_{k,j}\mathbb{E}_{\mathbf{B}_{k,j}}[\mathbf{B}_{k,j}^T \boldsymbol{\nu}_{k,j+1}], \\ \boldsymbol{\nu}_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j}, \mathbf{B}_{k,j}}[\bar{\mathbf{A}}_{k,j}^T \boldsymbol{\nu}_{k,j+1}] + \mathbf{Q}_j\mathbf{e}_{k,j}. \end{aligned} \quad (3.29)$$

See Appendix A.4 for a detailed derivation. Equation (3.29) is easier to implement, since the disturbances do not need to be estimated explicitly. The compensations $\delta \mathbf{u}_{k,j}$ are added to the total control inputs applied at iteration k . In an adaptive implementation, the feedback components of the update, $\mathbf{K}_{k,j}(\mathbf{e}_{k+1,j} - \mathbf{e}_{k,j})$, does not completely subtract the previous feedback controls $\mathbf{K}_{k-1,j}\mathbf{e}_{k,j}$ from the total control inputs, as the feedback matrices are also adapted over the iterations.

Typically ILC is used to feed the past errors along the trajectory (filtered and multiplied with a *learning* matrix) back to the system for the next trial as feedforward compensations. A well designed feedback controller, whenever available, is only used to reject nonrepeating disturbances and to stabilize the system in the time domain. The recursive implementation (3.29), on the other hand, readily provides and updates a feedback controller based on past performance. From here on, we will refer to the feedforward part of (3.29) as $\delta \mathbf{u}_{k,j}$, keeping the feedback control separate.

Computing the Expectations

The expectations appearing in (3.24) can be calculated given the covariances $\Sigma_{k,j}$ of the parameters,

$$\begin{aligned}
 \Phi_{k,j} &= \tilde{\Phi}_{k,j} + \mathbf{R}_j, \\
 \tilde{\Phi}_{k,j}^{a,b} &= \sum_{c=1}^n \sum_{d=1}^n \mathbf{P}_{k,j+1}^{c,d} \left(\mathbb{E}[\mathbf{B}_{k,j}^{c,a}] \mathbb{E}[\mathbf{B}_{k,j}^{d,b}] + \sigma(\mathbf{B}_{k,j}^{c,a}, \mathbf{B}_{k,j}^{d,b}) \right), \\
 \Psi_{k,j}^{a,b} &= \sum_{c=1}^n \sum_{d=1}^n \mathbf{P}_{k,j+1}^{c,d} \left(\mathbb{E}[\mathbf{B}_{k,j}^{c,a}] \mathbb{E}[\mathbf{A}_{k,j}^{d,b}] + \sigma(\mathbf{B}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}) \right), \\
 \mathbf{M}_{k,j}^{a,b} &= \sum_{c=1}^n \sum_{d=1}^n \mathbf{P}_{k,j+1}^{c,d} \left(\mathbb{E}[\mathbf{A}_{k,j}^{c,a}] \mathbb{E}[\mathbf{A}_{k,j}^{d,b}] + \sigma(\mathbf{A}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}) \right),
 \end{aligned} \tag{3.30}$$

where the upper indices a, b denote the corresponding entry of the matrix appearing on the left-hand side. The covariance matrices $\Sigma_{k,j}$ contain the scalar covariance terms $\sigma(\cdot)$ on the relevant entries, i.e.,

$$\begin{aligned}
 \sigma(\mathbf{A}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}) &= (\Sigma_{k,j})^{(a-1)n+c, (b-1)n+d}, \\
 \sigma(\mathbf{B}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}) &= (\Sigma_{k,j})^{n^2+(a-1)n+c, (b-1)n+d}, \\
 \sigma(\mathbf{B}_{k,j}^{c,a}, \mathbf{B}_{k,j}^{d,b}) &= (\Sigma_{k,j})^{n^2+(a-1)n+c, n^2+(b-1)n+d}.
 \end{aligned} \tag{3.31}$$

The indexes of $\mathbf{B}_{k,j}$ covariances start from n^2 since the model matrix parameters in (3.12) are vectorized starting from $\mathbf{A}_{k,j}$.

3.5 Online Implementation for Robot Table Tennis

In this section we algorithmically describe the recursive, adaptive and cautious *bayesILC* proposed in the last two sections in detail, with the extensions for an online robot learning application. We will consider tracking table tennis trajectories as our application of choice. The online learning algorithm is readily applicable to similar dynamic tasks with real-time constraints, such as throwing, catching skills in sports or fast, demanding manufacturing tasks.

The proposed ILC framework is summarized in Algorithm 3. Before entering the main loop (lines 7 – 16), the trajectory is executed with inverse dynamics and time-varying LQR feedback (line 5). The errors along the trajectory are filtered with a zero-phase filter (line 6). During the cautious ILC update the feedback control law as well as the feedforward control inputs are updated recursively (line 9). From the first iteration onwards, the means and the covariances of the model matrices are updated (line 14) before computing the feedforward input compensations $\delta \mathbf{u}_{k,j}$ and the feedback matrices $\mathbf{K}_{k,j}$. If the variant adaptation laws discussed in Section 3.3 are employed, it will be enough to store the means and covariances of the relevant model parameters. These parameters can then be transformed, as discussed before, to form the discrete-time model matrix means and covariances, which are used in the cautious ILC update (line 9).

Based on the forgetting factor λ , the model adaptation strikes a balance between the prior model parameter distribution and the data observed in iteration k . For the discrete LTV model

Algorithm 3 Recursive, adaptive and cautious *bayesILC*.

Require: $\mathbf{f}_{\text{nom}}, \mathbf{r}_j, \lambda, \epsilon > 0, \mathbf{Q}_j \succeq 0, \mathbf{R}_j \succ 0, \Sigma_{0,j} \succ 0$

- 1: Move to initial posture $\mathbf{q}_0 = \mathbf{r}_0, \dot{\mathbf{q}}_0 = \mathbf{0}$.
 - 2: Initialize $k = 1, \delta \mathbf{u}_{0,j} = \mathbf{0}, j = 1, \dots, N$
 - 3: Compute mean dyn. parameters $\mu_{0,j}$ by linearizing \mathbf{f}_{nom}
 - 4: Compute feedback $\mathbf{K}_{0,j} = \text{LQR}(\mathbf{Q}_j, \mathbf{R}_j, \mu_{0,j}, \Sigma_{0,j})$
 - 5: Execute with inv. dyn. \mathbf{u}_{IDM} and feedback $\mathbf{K}_{0,j}$
 - 6: Filter errors with a zero-phase filter (output: $\hat{\mathbf{e}}_{0,j}$)
 - 7: **repeat** ▷ ILC operation
 - 8: Compute error norm $\mathcal{J}_k = \left(\sum_{j=1}^N \hat{\mathbf{e}}_{k,j}^T \mathbf{Q}_j \hat{\mathbf{e}}_{k,j} \right)^{1/2}$
 - 9: Compute $\delta \mathbf{u}_{k,j}, \mathbf{K}_{k,j}$ recursively using (3.24) – (3.29)
 - 10: Update feedforward controls $\mathbf{u}_{k+1,j} = \mathbf{u}_{k,j} + \delta \mathbf{u}_{k,j}$
 - 11: Execute with $\mathbf{u}_{\text{IDM},j} + \mathbf{u}_{k+1,j}$ and feedback $\mathbf{K}_{k,j}$
 - 12: Observe errors $\mathbf{e}_{k,j} = \mathbf{x}_{k,j} - \mathbf{r}_j$
 - 13: Filter errors with a zero-phase filter (output: $\hat{\mathbf{e}}_{k,j}$)
 - 14: Update model $\mu_{k,j}, \Sigma_{k,j}$ using (3.18)
 - 15: $k \leftarrow k + 1$
 - 16: **until** $\mathcal{J}_k < \epsilon$
-

and the link parameter adaptation, the data used is $\mathbf{y}_{k,j} = \hat{\mathbf{e}}_{k,j+1} - \hat{\mathbf{e}}_{k-1,j+1}$. If continuous model matrix adaptation is performed, the data will instead be the smoothened joint acceleration differences, see (3.19). We discuss the effects of the forgetting factor and the different model adaptation strategies in more detail in Section 3.6.

The practitioner, wary of the model inaccuracies, can increase robustness and ensure stability by setting large diagonal terms for the initial covariance of model uncertainty, $\Sigma_{0,j} = \gamma \mathbf{I}, \gamma \gg 1, j = 1, \dots, N$. Moreover, setting large covariances initially helps to observe the inaccuracies of the model and the noise statistics. The covariance will be suitably decreased over the iterations, as adaptation (3.18) updates the linear models. Observing the noise statistics over the iterations can further help in the design of a good zero-phase filter to reject noise. Without accurate smoothing, adaptive ILC approaches run the risk of picking up noise in the adapted model matrices, which are then used in the control input update (in our case, in equation (3.29)). This can hinder the control performance, hence we advice caution in the design of a smoothening filter.

The proposed update law takes advantage of the learning efficiency and computational advantages of model-based recursive ILC while being cautious with respect to model mismatch. The computational complexity of the recursive update is $\mathcal{O}(Nn^3)$ as opposed to batch norm-optimal ILC, where the batch pseudoinverse operation typically incurs $\mathcal{O}(N^3n^3)$ complexity. The batch model-based implementation using the *lifted-vector form* [43] inverts the input-to-output matrix \mathbf{F} ,

$$\begin{aligned} \mathbf{U}_{k+1} &= \mathbf{U}_k - \mathbf{F}^\dagger \mathbf{E}_k, \\ \mathbf{E}_k &= \left(\mathbf{e}_{k,1}^T, \mathbf{e}_{k,2}^T, \dots, \mathbf{e}_{k,N}^T \right)^T, \end{aligned} \tag{3.32}$$

Algorithm 4 ILC improving execution of robot table tennis hitting movements online.

Require: \mathbf{q}_0 , \mathbf{f}_{ball} , $\text{bayesILC}(\dots)$ (see Algorithm3)

- 1: Move to initial posture \mathbf{q}_0 , $\dot{\mathbf{q}}_0 = \mathbf{0}$.
 - 2: Predict ball trajectory \mathbf{b}_j using \mathbf{f}_{ball}
 - 3: Compute trajectory \mathbf{r}_j given \mathbf{q}_0 and \mathbf{b}_j , $j = 1, \dots, N$
 - 4: Setup bayesILC (lines 2 – 4)
 - 5: **repeat** ▷ fixed ballgun throws balls at a constant rate
 - 6: Execute strike with \mathbf{u}_{ILC} and LQR feedback \mathbf{K}
 - 7: Return to \mathbf{q}_0 with high-gain PD control and linear traj.
 - 8: Update \mathbf{u}_{ILC} and \mathbf{K} with bayesILC (lines 9 – 14)
 - 9: **until** ballgun is moved
-

where the submatrices of the input-to-output matrix \mathbf{F} are

$$\mathbf{F}_{(i,j)} = \begin{cases} \mathbf{A}_{i-1} \dots \mathbf{A}_j \mathbf{B}_{j-1}, & j < i, \\ \mathbf{B}_{j-1}, & j = i, \\ \mathbf{0}, & j > i. \end{cases} \quad (3.33)$$

The condition of the lifted model matrix (3.33) grows exponentially with N and inverting it quickly becomes numerically unstable.

Implementation for Tracking Table Tennis Trajectories

The online learning framework for robot table tennis is described in Algorithm 4. Whenever a ball is initialized from a fixed ballgun with constant settings, located at \mathbf{b}_0 , the trajectory generation framework will compute a particular striking trajectory (lines 2–3) to intercept and hit the ball in real time. See Chapter 2 for the trajectory generation pipeline. ILC can then be initialized (line 4) by linearizing the dynamics model \mathbf{f}_{nom} around the computed trajectory points \mathbf{r}_j , $j = 1, \dots, N$. ILC needs to be initialized only once, as long as the computed trajectory is capable of returning the ball to the opponent’s court. The approximately 8cm radius of the racket can cover for the inconstancy of the ballgun up to a certain degree.

Whenever the striking trajectory is executed (line 6), a returning trajectory will bring the arm back (line 7) from the current state to the fixed initial posture, \mathbf{q}_0 . The returning trajectory can be as simple as a linear trajectory in the joint space. The *consistency* provided by the fixed ballgun in our setup, shown in Figure 4.1, allows us to use ILC to track invariant trajectories over the iterations.

For a good performance in table tennis, the striking parts of these hitting movements need to be tracked accurately. The strikes are initially tracked with computed-torque inverse dynamics feedforward control commands and the additional LQR feedback. The feedback law is computed for this purpose by linearizing the nominal dynamics model around the striking part of the reference trajectory. After a strike is completed, feedback will switch to PD-gains for the returning trajectory and the arm will come back close to \mathbf{q}_0 . Learning with ILC can then take place (line 8) while waiting for another incoming table tennis ball.

The striking trajectory in table tennis is only an intermediary and does not need to be precisely tracked for a successful performance. In general, for hitting and catching tasks, the task performance depends critically on reaching the desired joint positions and velocities at the final time. A good performance along the trajectories is a means to this desired end: if feedback keeps the system stable around the trajectories, and the (linearized) models are reasonably valid around the trajectories, convergence to desirable performance levels can be rapid.

Coping with Varying Initial Conditions

Execution errors in tracking the reference trajectory (including the returning segment) prevents the robot from initializing in each iteration at the same state. Putting very high feedback gains on the returning trajectory or waiting long enough may suffice to initialize the system close to desired initial conditions, but in some occasions, none of these options may be desirable or available. For example, a robot *practicing* table tennis with a fixed ballgun running at a fixed rate, may not have time to initialize its desired posture accurately.

Starting from varying initial conditions $\mathbf{x}_{k,0} = [\mathbf{q}_{k,0}^\top, \dot{\mathbf{q}}_{k,0}^\top]^\top$ one can consider updating the hitting movement \mathbf{r}_j to take the robot to the same hitting state. For such online updating of trajectories, the invariant trajectory parameters \mathbf{p} can be used to generate the trajectory from the current joint values. The reference control inputs \mathbf{u}_{IDM} can then be recomputed based on the nominal inverse dynamics model. With this correction the total feedforward control commands \mathbf{u}_{ILC} at iteration $k + 1$ are re-computed as

$$\mathbf{u}_{\text{ILC},j} = \mathbf{u}_{k+1,j} + \mathbf{u}_{\text{IDM},j}(\tilde{\mathbf{r}}_j) - \mathbf{u}_{\text{IDM},j}(\mathbf{r}_j), \quad (3.34)$$

where $\tilde{\mathbf{r}}_j$ is the updated trajectory starting from the perturbed initial state $\mathbf{x}_0 + \delta\mathbf{x}_{k,0}$. Using this simple adjustment (3.34), the stability of the learning performance can be greatly improved.

3.6 Evaluations and Experiments

In this section, we demonstrate the effectiveness of the ILC algorithm *bayesILC* presented in Algorithm 3 and described in detail in Section 3.5 in the context of tracking table tennis trajectories. We validate the proposed learning control law first in extensive simulations with linear and nonlinear models. In the last section we show real robot experiments with two seven degree of freedom Barrett WAM arms for tracking table tennis striking movements.

3.6.1 Verification on Toy Problems

Stability is an important issue in the implementation of different learning controllers in real robot tasks. As a result, we setup extensive simulation experiments to validate the stability and robustness of our learning approach. We also discuss in detail the advantages of the recursive formulation over the batch pseudo-inverse ILC (3.32).

Random Linear Models

We generate here random linear models and random trajectories drawn from Gaussian Processes (GP) with squared exponential kernels. More specifically, the elements of the linear time-varying

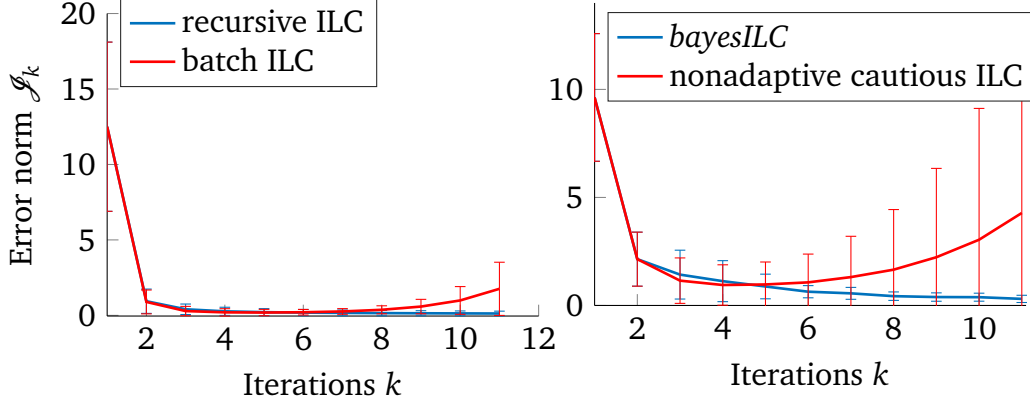


Figure 3.4: ILC in recursive form is evaluated on random linear time-varying (LTV) systems. The Frobenius norm of the trajectory deviations, \mathcal{J}_k , is plotted over the iterations k . Results are averaged over ten experiments, where for each experiment, trajectories, nominal models and actual models are drawn randomly from Gaussian Processes. The performance of the batch pseudo-inverse ILC (3.32) is shown in the red line. Numerical stability issues prevent it from stabilizing at steady state error, whereas recursive ILC (blue line) converges stably. If the model mismatch is increased, at some point, recursive ILC also diverges. Applying caution without adaptation is not enough to converge to steady state error. Cautious *and* adaptive *bayesILC*, on the other hand, applying the updates (3.14) and (3.29) iteratively, is very effective and shows a stably convergent behaviour.

(LTV) model matrices $\mathbf{A}_j, \mathbf{B}_j$ are drawn from $(n + m)n$ uncorrelated GPs. The hyperparameters (scale, noise and smoothness parameters) of these GPs are drawn independently from normal distributions with fixed means and variances. Moreover, random perturbations of these models (drawn the same way from $(n + m)n$ uncorrelated GP's) are generated to construct nominal models. Using the proposed random disturbance generation scheme, we can average the results and construct error bars for different ILC algorithms.

The performance of the recursive implementation (i.e., Equation (3.29) with zero covariances and no adaptation) is shown in Figure 3.4 on the left-hand side, where the results are averaged over ten different trajectories and models. The dimensions of the models are $n = 2, m = 2$, and the horizon size is set to $N = 120$. For the LQR and ILC calculations, $\mathbf{R} = 10^{-6}\mathbf{I}$ and the weighting matrix \mathbf{Q} was set to the identity. In this case, the batch model-based implementation using the pseudo-inverse (3.32) is not stable at all without feedback. Applying LQR feedback and adding current iteration ILC in Figure 3.4 improves the performance (red line in Figure 3.4), but numerical issues (i.e., large condition number) in inverting the large model matrix \mathbf{F} in lifted form (3.33) prevents it from stabilizing at steady state error. Tracking performance throughout the experiments is measured with respect to the Frobenius norm of the deviations $\mathbf{e}_{k,j}$, denoted as \mathcal{J}_k .

For the simulation results in Figure 3.4, the spectral norm of the difference between the nominal and the actual models are each set to $\alpha\sigma_{\min}(\mathbf{F})$ where $\alpha = 100$. Increasing α further increases the probability that the model-based ILC is not monotonically convergent for some trial. For example, one can observe *asymptotically* but not *monotonically* convergent ILC

behaviour when setting $\alpha = 990$ for a particular model and trajectory shown in Figure 3.2. Increasing α further can prevent even asymptotic stability.

Especially in these cases of high model mismatch, the proposed adaptive and cautious *bayesILC* offers a stable and convergent ILC behaviour. In Figure 3.4 on the right-hand side, we consider the case where $\alpha = 1000$. Recursive ILC that is also cautious does not show a stable convergent behaviour, whereas recursive ILC that is not cautious (i.e., covariance of the LTV matrices are zero) is not stable at all. Cautious *and* adaptive *bayesILC*, on the other hand, using LBR ($\lambda = 1.0$) to update the discrete-time LTV matrices $\mathbf{A}_{k,j}, \mathbf{B}_{k,j}$, shows a monotonic learning performance. The results are again averaged over ten different models and ten trajectories. For LBR, the initial covariances in (3.14) are set to $\Sigma_{0,j} = \gamma \mathbf{I}$ for all $j = 1, \dots, N$, where $\gamma = 10^4$ and the noise covariance is $\sigma^2 = 1$. Changing the exponent of the initial covariance, or reducing the forgetting factor λ in this case, can lead to a reduced or unstable learning performance.

Gaussian Process Dynamics

The performance of the proposed algorithm *bayesILC* is evaluated next over random nonlinear models. In these set of experiments, we sample the states from n uncorrelated GPs with squared exponential kernels and random linear mean functions. The hyperparameters of these GPs are randomized as before. By sampling from such random nonlinear models, we can test the proposed algorithm under nonlinear uncertainties and noisy outputs. The *actual* model is simulated as follows:

1. Random reference control inputs $\mathbf{v}_j \in \mathbb{R}^m, j = 1, \dots, N$ are drawn K times from m *control* GPs.
2. n *oracle* GPs are used to sample $\mathbf{f}(\mathbf{x}_j, \mathbf{v}_j)$ and the generated dynamics is integrated (starting from zero initial conditions) using forward Euler, $dt = 1/N$, to form K trajectories. The GPs are conditioned during this process on the generated states \mathbf{x}_j and inputs \mathbf{v}_j .

These n oracle GPs constitute the *actual* but unknown nonlinear dynamics model. Nominal models can be easily generated by using the predictions of the oracle GPs at a subset of the state space. The construction of a *nominal* model is described in detail below:

1. Another set of control inputs $\mathbf{u}_j, j = 1, \dots, N$ are drawn from the *control* GPs, as before.
2. The mean predictions $\mathbf{f}(\mathbf{x}_j, \mathbf{u}_j)$ of the oracle GPs at \mathbf{u}_j are used to evolve these control inputs (as in step 2 of the actual model).
3. The n separate *model* GPs (with same hyperparameters as the oracle) are conditioned on the resulting trajectory, i.e., the input pairs $(\mathbf{x}_j, \mathbf{u}_j)$ and the outputs $\mathbf{f}_j = (\mathbf{x}_{j+1} - \mathbf{x}_j)/dt$ for each time step $j = 1, \dots, N$.
4. The mean derivative of the model GPs are calculated analytically (using the kernel derivatives). Discretized time-varying matrices $\mathbf{A}_j, \mathbf{B}_j$ and their variances $\Sigma_{0,j}$ are constructed for each $j = 1, \dots, N$, based on the mean and variance of the GP derivatives.

By sampling $K = 20$ trajectories for the conditioning of oracle GPs, we can cover a significant part of the state space in $n = 2$ dimensions. For each ILC iteration thereafter, the mean predictions are used as in step (2) to evolve the trajectory, but without further conditioning of the

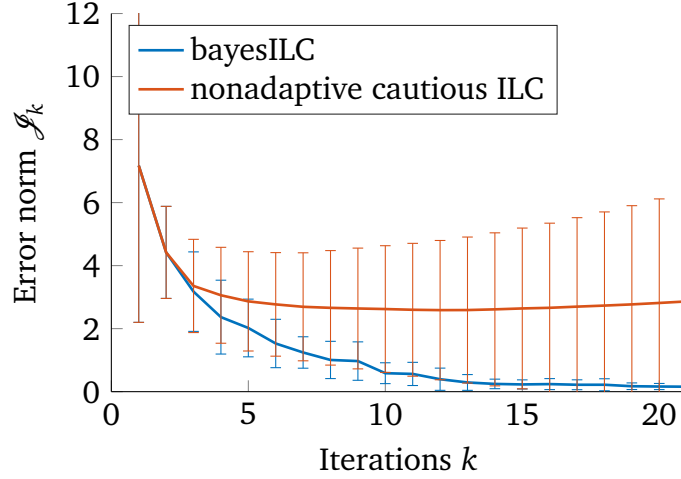


Figure 3.5: The proposed ILC algorithm is evaluated on random nonlinear systems. The Frobenius norm of the trajectory deviations, \mathcal{J}_k , is plotted over the iterations k . Results are averaged over ten experiments, where for each experiment, trajectories and dynamics along these trajectories are drawn from Gaussian Processes. Recursive ILC that is not cautious shows an unstable behaviour, and adding adaptation without caution is also not stable (both not shown in the Figure). Purely cautious ILC (red line) is divergent for some of the trajectories. Cautious *and* adaptive *bayesILC*, on the other hand (blue line), shows a stable convergent learning performance.

model GPs. Instead, adaptation is performed as before with LBR, replacing the steps (3–4). We can thus avoid the expensive online GP training.

Figure 3.5 shows the learning performance for a horizon size of $N = 20$. The dimensions of the system is same as before, $n = 2, m = 2$ and $\mathbf{R} = 10^{-6}\mathbf{I}$, $\mathbf{Q} = \mathbf{I}$. The results are averaged again over ten experiments. In this nonlinear setting, the recursive ILC that is not cautious shows an unstable behaviour (not shown in Figure 3.5). Adaptive but not cautious ILC is also unstable (also not shown). Cautious but not adaptive ILC is not stable for some trajectories and can diverge (red line). Cautious *and* adaptive *bayesILC*, on the other hand (blue line), using LBR to update the discrete-time LTV matrices, shows again a stable convergent learning performance, improving over the purely cautious ILC. For LBR, the initial covariances in (3.14) are again set to $\gamma = 10^4$ times the identity and the noise covariance is $\sigma^2 = 1$. The best performance is reached when the forgetting factor is set to $\lambda = 0.9$. As before, changing the exponent of the initial covariance, or the forgetting factor, can lead to a reduced or unstable learning performance.

Barrett WAM Model

We next test ILC on striking movements (see Chapter 2, Section 2.4 for the Focused Player) for a seven degree of freedom Barrett WAM simulation model. In the simulations, the robot is started from a fixed initial state \mathbf{q}_0 . The initial posture is chosen from one of the center, right-hand side or left-hand side resting postures of the robot. The striking parameters (2.8) are then optimized, based on an incoming table tennis ball with a randomly chosen incoming position and velocity. The link parameters of the Barrett WAM forward dynamics model used to simulate actual trajectories are perturbed randomly to construct nominal models for ILC. The

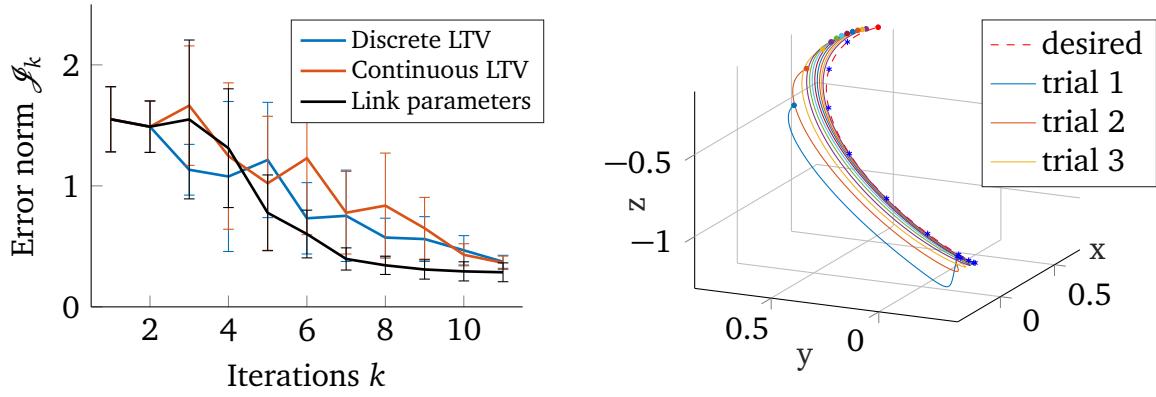


Figure 3.6: The performance of the adaptive and cautious ILC algorithm *bayesILC* on the simulated Barrett WAM model is shown on the left-hand side. The Frobenius norm of the trajectory deviations, J_k , is plotted over the iterations k . The results are averaged over ten different strikes and three different initial postures. Three different adaptation laws are considered, adaptation of discrete-time and continuous-time LTV models are shown in blue and red, respectively, while the adaptation of link parameters is shown in black. Forgetting factor was set to $\lambda = 0.8$ for all of the adaptation laws. One of the desired trajectories, shown in dashed red on the right-hand side, is tracked very closely in the final iteration. The blue markers correspond to the time profile of the motion, which are drawn uniformly spaced, one for each 80 milliseconds. The final hitting positions reached are shown as filled circles.

linearization procedure described in Section 3.2 produces LTV nominal models that can be used by ILC to reduce the deviations from the desired (fixed) striking movement over the iterations.

The randomization during the optimization guarantees that a variety of hitting movements are tracked throughout the experiments. The performance of the proposed ILC approach *bayesILC* with three different adaptation laws is then evaluated over the striking segment of the optimized (striking and returning) trajectories. The convergence results are averaged over ten such striking movements, as shown in Figure 3.6. The adaptation of discrete-time and continuous-time LTV models are shown in blue and red, respectively, while the adaptation of link parameters is shown in black. Forgetting factor was set to $\lambda = 0.8$ for all of the adaptation laws. Initial covariances are set to $\Sigma_{0,j} = 10^4 \mathbf{I}$ for continuous and discrete-time LTV model adaptation laws, while for link parameters, the initial covariances are $\Sigma_{0,j} = 10^{10} \mathbf{I}$. The weights of the cautious ILC update (3.29) is set to $\mathbf{R} = 10^{-2} \mathbf{I}$, $\mathbf{Q} = \mathbf{I}$.

After updating the link parameter means and variances, we use an auto-differentiation tool (ADOL-C library in C++) together with sampling to approximate the distribution of forward dynamics (3.3) derivatives $\mathbf{A}_{k,j}$, $\mathbf{B}_{k,j}$. More specifically, the forward dynamics is differentiated (with respect to joint positions, velocities and control inputs) at 100 link parameter samples drawn from the posterior distribution (i.e., normal distribution with means and variances given by (3.14)) online. This sampling procedure generates a reasonable approximation of posterior derivative means and variances.

In table tennis, if the robot arm follows the assigned reference trajectory precisely it will hit the ball with a desired velocity at the desired time. We can see on the right-hand side of Figure 3.6 that an initial attempt (blue curve) falls short of the reference trajectories (dashed curve). The percentage of the balls that are returned to the opponent's court are close to zero.

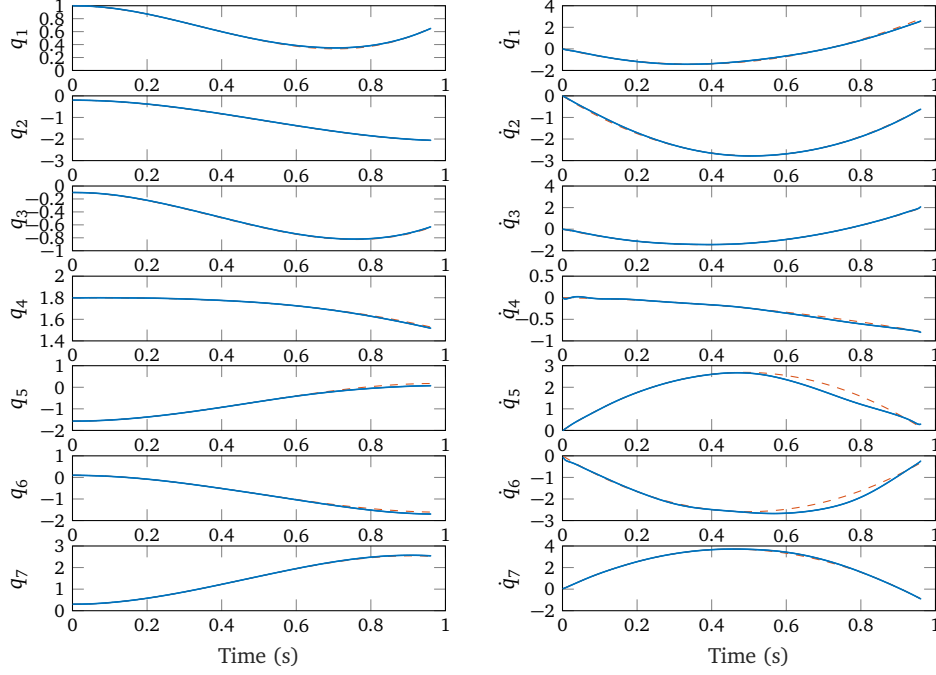


Figure 3.7: Joint trajectories for a hitting movement on the Barrett WAM model. The reference trajectories, shown in dashed red, are tracked very closely with ILC in the final iteration, shown in blue.

ILC then modifies the control inputs to compensate for the modeling errors. In the last attempt the reference trajectory is executed almost perfectly. The accuracy of the table tennis task increases to %95, on average. Figure 3.7 shows the adjusted control inputs for one striking movement.

The recursive ILC (without adaptation or caution) is convergent for some of the hitting movements in Figure 3.6. However, similar to the previous simulation examples, the recursive form of the ILC update, depending on the accuracy of the model along the trajectories, can fail to converge for some trajectories (not shown in the Figure). The proposed recursive, adaptive and cautious algorithm *bayesILC*, with the three adaptation laws shown in Figure 3.6, shows a better and faster convergence, for a variety of trajectories.

The ILC experiments shown in Figures 3.6–3.7 reset the initial posture always to the same desired posture \mathbf{q}_0 . Next, we consider non-repetitive disturbances around the desired initial posture. This would mean, physically, that the robot is not initialized accurately around the resting posture.

Comparisons to the baseline (black line) in Figure 3.8 illustrate the additional robustness whenever the trajectory adaptation (3.34) is employed. We adapt the metric for this comparison according to the task: the costs indicated are the *final* costs (for hitting the incoming ball at the desired joint positions with desired joint velocities), not the full costs incurred along the reference trajectory. Note especially the faster convergence and increased accuracy of the proposed method with the reference trajectory and input adaptation (blue line). More robust performance is obtained by adapting the trajectories \mathbf{r}_j and $\mathbf{u}_{\text{IDM},j}$, which, in addition to performing better, shows much lower variance compared to the baseline.

In practice trusting the model too much at the beginning of the trajectory leads to the amplification of initial errors. Nonrepetitive starting postures violate the initial condition assumption

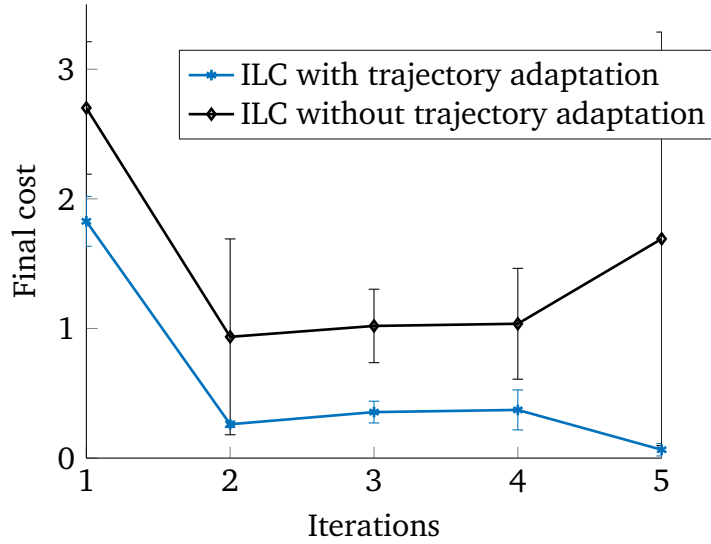


Figure 3.8: Simulation results illustrating the additional robustness to varying initial conditions whenever the trajectories (states and control references) are adapted according to (3.34) (blue line). Note the unstable performance of ILC without such adaptation (black line), which keeps the references \mathbf{r}_j and the inverse dynamics inputs $\mathbf{u}_{\text{IDM},j}$ fixed.

typical of standard ILC updates. In this case, the feedback matrices $\mathbf{K}_{k,j}$, as opposed to the feed-forward input updates $\delta \mathbf{u}_{k,j}$, play a bigger role in the learning stability at the beginning of the trajectories, $j \ll N$.

3.6.2 Real Robot Table Tennis

Finally we perform experiments on our robotic table tennis platform, see Figure 3.10, where two seven degree of freedom (DoF) cable-driven, torque-controlled Barrett WAM arms (*Ping* and *Pong*) are hanging from the ceiling. The custom made Barrett WAM arms are capable of high speeds and accelerations (approx. up to 10m/s^2 in task space). Standard size rackets (16 cm diameter) are mounted on the end-effector of the arms as can be seen in Figure 3.10. A vision system consisting of four cameras hanging from the ceiling around each corner of the table is used for tracking the ball [41]. A ball launcher (see Figure 4.1) is available to throw balls accurately to a fixed position inside the workspace of the robots. The incoming ball arrives with low-variability in desired positions and higher-variability in ball velocities. The whole area to be covered amounts to about 1 m^2 circular region surrounding an initial centered posture of the robots.

The realistic simulation environment SL [38] acts as both a simulator and as a real-time interface to the Barrett WAMs in our experiments. The initial positioning is given by a PD controller with high gains on the shoulder joints, which is then toggled off during the experiments with the striking movements, as summarized in Algorithm 4. The high-gain PD controller used to initialize the robots was also tested for tracking the striking movements, see Figure 3.9. When ILC is applied on top of the PD controller, the learning quickly stagnates, leading to oscillations in some of the joints. Instead, a low-gain LQR feedback law is computed for the striking part of the movement with a linearized nominal dynamics model (3.7). The weighting matrices for

this purpose are set to identity, $\mathbf{Q} = \mathbf{I}$, and the constant penalty matrix is chosen as $\mathbf{R} = 0.05\mathbf{I}$. Decreasing the scaling of the penalty matrix to 0.03 causes oscillations in the elbow joint, indicating that the nominal model is not very accurate. At the cost of larger initial error, we suggest increasing the input penalties \mathbf{R} to improve the stability of ILC in other high degrees-of-freedom robotics applications.

After the visual system outputs a ball estimate, a ball model can be used along with an Extended Kalman Filter to predict a ball trajectory. The ball model accounts for some of the bouncing behavior of the ball and air drag effects. If the predicted ball trajectory coincides with the workspace of the robot, the motion planning system has to then come up with a trajectory that specifies how, where and when to intercept the incoming ball. Desired Cartesian position, velocity and orientations of the racket at the hitting time T impose constraints on the seven joint angles and seven joint velocities of the robot arm at T . Along with the desired hitting time T (or the time until impact), these fifteen parameters are used to generate third-order joint space polynomials. These movements can be optimized online in 20–30 milliseconds [2], or loaded from a lookup table. In the ILC experiments, the parameters in the lookup table are used without interpolation, to make sure that the same trajectory can be used for balls deviating slightly from their stored values. We make sure that the lookup table is dense enough and that the ballgun is fixed.

Some examples of the generated trajectories are shown in Figure 3.10. After a strike, a linear joint trajectory is computed that will take the robots from the current state to the resting posture in $T_{\text{rest}} = 1.0$ seconds. PD feedback control is turned on again for this returning part of the trajectory. When the returning trajectory is executed, SL main thread running the inverse dynamics computations will continue to keep the arms stable around the resting posture, while another thread is detached to run the ILC update⁴. The ILC loop terminates successfully

⁴ Code is available in the public repository <https://gitlab.tuebingen.mpg.de/okoc/learning-control> along with the test scripts used to generate the plots in the previous subsections.

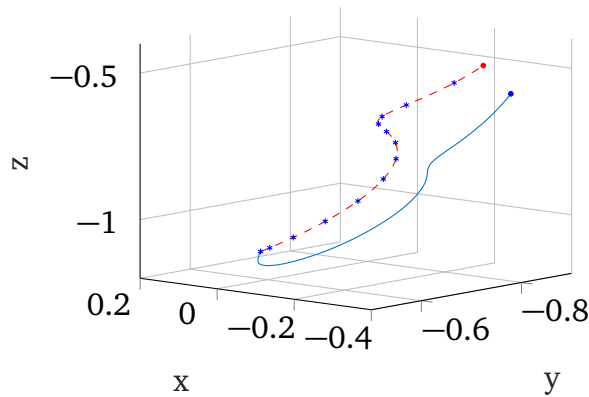


Figure 3.9: An example of a striking movement for real robot table tennis is shown in red. The blue markers correspond to the time profile of the motion, which are drawn uniformly, one for each 80 milliseconds. Executing this movement well with the Barrett WAM will lead to a good hit. Control errors in tracking lead to a poor hitting performance, shown in blue. The filled circles are the final reached hitting positions. High-gain PD feedback was used to track the reference in this real robot example. The tracking errors can be decreased efficiently and stably by applying the proposed recursive, cautious and adaptive ILC update *bayesILC*.

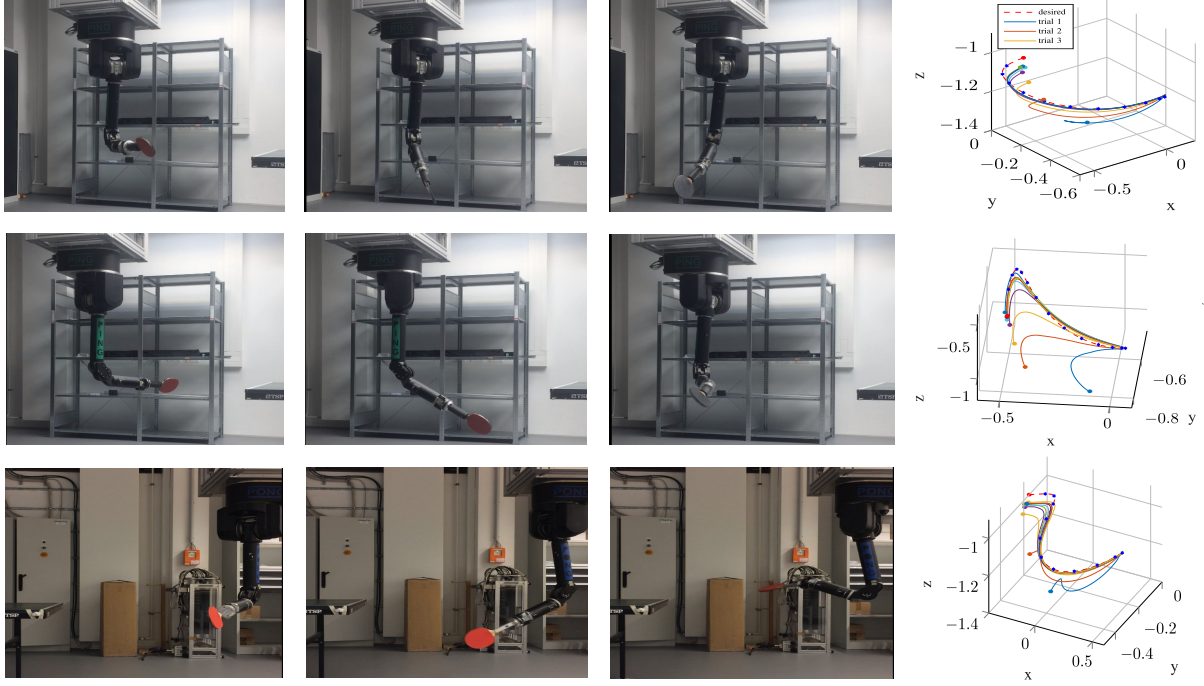


Figure 3.10: Two Barrett WAMs (a.k.a. *Ping* and *Pong*) are initialized in our experiments in three different starting postures. We make controlled experiments with a simulated ball-gun, and generate many different hitting movements, three of them are shown in the above images. The proposed algorithm *bayes/ILC* leads to an efficient and stable learning approach for tracking these hitting movements. The right-hand side starting posture for the robot *Ping* can be seen on the upper left image. Initially, before learning with ILC starts, *Ping* performs poorly, and the hitting posture of the robot is shown in the upper central image. After five iterations, the hitting posture is corrected significantly as shown in the upper right image. Similarly, the central images show the operation of the ILC for another trajectory, where the starting posture for *Ping* is fixed on the left-hand side of the robot. On the bottom images, an ILC performance is shown for the robot *Pong*. The three plots on the right-hand side show the Cartesian trajectories corresponding to the ILC iterations. The reference trajectories are shown in dashed red, and the final hitting positions reached are shown as filled circles.

whenever the computed feedforward updates are within the respective torque limits. After a successful termination, if the actual posture is within 0.1 radians distance of the resting posture, the LQR feedback will be turned on again and the robots will start moving to track the same striking motion.

We use a simulated ball to make more controlled experiments, focusing on the control aspect in more detail. If the striking robot movements are executed accurately, then the ball in simulation will be returned close to a desired position on the opponent's court. At different points in time we have identified three different sets of link parameters for rigid body dynamics. We can use these parameterizations of rigid body dynamics as potential nominal models to kick-start the learning process. We tested these nominal models first in slowed down hitting movements, where a slow down rate of two means that the number of trajectory points double while the hitting time is held fixed. Cutting down the trajectories to an initial subset of the

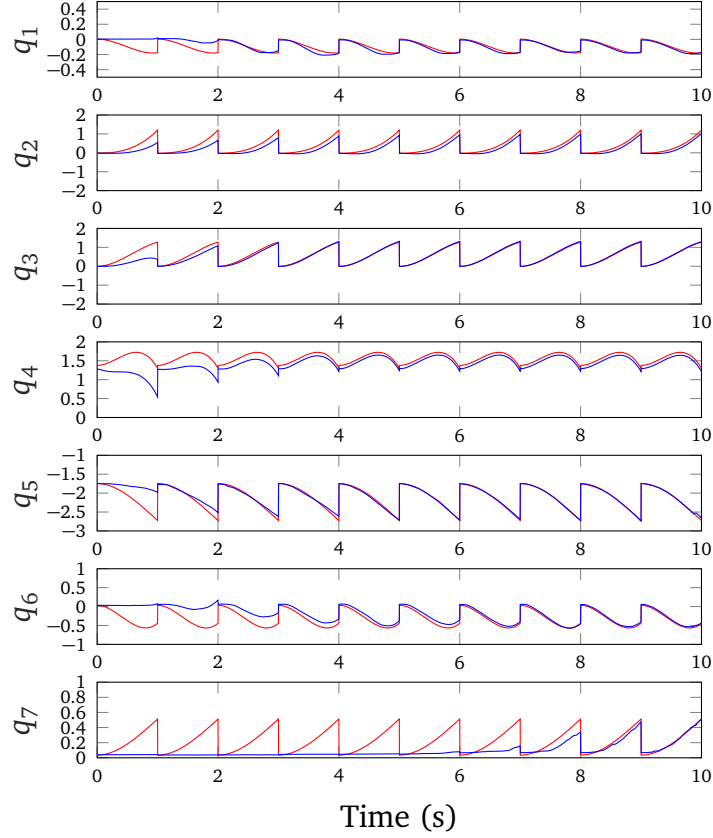


Figure 3.11: Robot experiment results for cautious and adaptive *bayesILC*, shown for a particular reference trajectory. The ten iteration results are concatenated for convenience. The desired joint trajectories correspond to a hitting movement on the Barrett WAM. The reference trajectories, shown in red, are tracked very closely with ILC in the final iteration, shown in blue. Final cost goes down to 0.20 in the last iteration.

movement to restrict potential instabilities, or initial masking of some of the joints during ILC updates, are other techniques that we have employed to evaluate these nominal models in a careful manner. Of the three models, only one of them was suitable for the *local* learning that ILC provides. This model is further adapted with the proposed *bayesILC* algorithm in order to improve the tracking of the striking movements. Adaptation of the trajectories \mathbf{r}_j and the nominal inputs $\mathbf{u}_{\text{IDM},j}$ was additionally performed on top of ILC, to stabilize the learning process, since an accurate initialization of the joints (especially on the wrist and the elbow) was not possible with the Barrett WAMs.

We have compared *bayesILC* to two other ILC methods: batch ILC (3.32) and ILC with proportional and derivative (PD) feedback (with constant p, d values). PD type ILC with constant p or d values is often too simplistic, and did not yield any improvement in our setup, even after tuning the p, d values. Batch ILC was tested with ten times downsampled trajectories, with adjustable learning rates. We have found batch ILC to be inferior to the recursive ILC when

tested over multiple trajectories (slowed down and cut versions included)⁵. Recursive ILC without any adaptation is monotonically convergent on average for about five iterations, bringing the root mean squared (RMS) tracking error from about 0.80 to 0.40 on average. Repeating the trajectories for five more iterations, we note that the tracking error starts increasing slightly due to introduced oscillations in some of the joints. Introducing adaptation with recursive and cautious ILC (i.e., the proposed approach *bayesILC*) we can decrease the tracking error further, to about 0.20 monotonically in five more iterations. This enables a return accuracy of 40% of the simulated balls to the opponent’s court.

The proposed update law *bayesILC* evaluated above adapts the discrete-time LTV models with a forgetting factor of $\lambda = 0.8$. This value was chosen experimentally, and could be optimized, e.g., using a dataset of previous ILC performances. The same parameter values are chosen for the initial covariances as in the simulation experiments with the Barrett WAM. Adapting the continuous LTV models, when the trajectories are smoothened suitably with a zero-phase filter, leads to faster updates with similar improvements in tracking performance. Using the online adaptation of the link parameters on the other hand, leads to poorer convergence in tracking for some of the joints (most notably, the elbow). This fact leads us to suspect that the rigid body dynamics model underfits, i.e., the mismatch for our Barrett WAMs is not purely parametric in nature. We see that the final cost (as 2-norm of deviations from desired joint hitting positions and velocities) drops down from 1.70 to 0.20 for *bayesILC* when the LTV model matrices are adapted directly. After performing ten more iterations, the percentage of balls successfully returned to the opponent’s court increases from 40% to about 60% on average⁶.

3.7 Conclusion and Future Work

In this chapter we presented a novel Iterative Learning Control (ILC) algorithm that is recursive, cautious and adaptive at the same time. The closed-form update law (3.24) that was presented derives from the adaptive dual control literature and is sometimes referred to as *passive learning* [14]. The algorithm was then recast in a more efficient form (derived in Appendix A.4) which does not require the estimation of disturbances and can be implemented as a recursive ILC update. The update law makes it easy to introduce caution with respect to modelling uncertainties and online adaptation of the linearized model matrices. Unlike typical ILC updates, feedback matrices for the tracking of striking trajectories are adapted as well, which are useful for rejecting noise and varying initial conditions. We believe that the introduced ILC update yields a principled approach to adapt the models, as well as their regularizer, based on data.

The proposed algorithm *bayesILC* was evaluated in different simulations of increasing complexity. Finally in the last subsection we have presented real robot experiments on our robotic table tennis setup with two Barrett WAMs, see Figure 3.10. It was shown that the proposed approach leads to an efficient way to learn to track hitting movements online. Hitting movements throughout the experiments are generated in the joint space of the robots and enable them to

⁵ For batch pseduoinverse-based ILC, inversion of the model matrices (3.7) around the *unstable* hitting trajectory causes instability, which is alleviated by providing an additional *current iteration ILC* (CI) [43]. CI adds the current iteration k ’s feedback errors to the feedforward compensations for the next iteration $k + 1$. As in our preliminary experiments with the Barrett WAM [66], we have applied CI in addition to stabilize a downsampled version of batch model-based ILC.

⁶ A video showing some example ILC performances for the two robots is available online: <https://youtu.be/27vHoLBwLoM>.

execute optimal striking motions. Control inputs, as well as a time-varying feedback law, are updated after each trial by using the model-based update rule that considers the deviations from the striking trajectory. After the trajectories are executed, the deviations can be used to adapt the model parameter means and variances using Linear Bayesian Regression (LBR). A forgetting factor was considered in addition to make adaptation more flexible. An adaptation of the reference trajectories as well as the nominal inputs was considered on top of *bayesILC* to render the method more effective and stable for initial posture stabilization errors.

Although we have shown a stable and efficient way to learn to track references with ILC, we have not analyzed its generalization to arbitrary trajectories. In our table tennis setup, we are making progress to having the two robots play against each other. Generalization capacity would play an important role in extending the average game duration between the robots, as the trajectories during the table tennis matches would be generated online [2] according to the state of the game. We believe that in the case where the trajectories are changing, generalizing the learned control commands can be achieved by compressing them to a lower-dimensional input space (i.e., parameters). Learned feedforward commands could be projected to a parameterized feedback matrix, the parameters of which could represent the invariants between the trajectories. An efficient and stable implementation of such parameterizations will be the focus of our future work.

4 Learning to Serve: an Experimental Study

Humans are good at using their bodies to great effect, taking advantage of their muscular structure and soft but flexible actuation. Much of dexterous manipulation, or dynamic movement generation reflects this awareness of the human body. When teaching the robots to achieve similar tasks autonomously, however, we inevitably impose and transfer our biases to the robot. This problem of *embodiment* can cripple the execution, possibly also preventing the robots from taking advantage of their kinematics structure and actuation mechanisms.

4.1 Introduction

In dynamic games like table tennis, we can easily observe humans taking utmost advantage of their bodies and pushing it to its maximum, i.e., optimizing their output bearing in mind their kinematic and dynamic limits. Table tennis serves, for instance, incorporate flicks (very fast accelerations of the wrist) that are designed to give an unsuspected spin and motion profile to the ball. Teaching such movements to the robots in a learning from demonstrations framework using kinesthetic teach-in, where the robot joint movements are recorded, suffers in particular from two drawbacks. Firstly, during the shown movement, as discussed above, the human is unable to move the shoulder joints of the robot adequately, which could potentially be used by the robot to great effect. Secondly, the fast movements of the wrists may not be tracked accurately by the robot, which is the case for the cable-driven seven degree of freedom (DoF) Barrett WAM arm, see Figure 4.1.

In this chapter, we explore different learning from demonstrations (LfD) approaches to compensate for the execution and transfer deficiencies resulting from the demonstrated serves. The demonstrations are acquired and the movement primitives are trained in the joint-space of the robot, using kinesthetic teach-in, where the movements of the robot are recorded using the joint-level sensors. The initial policy or the movement template, extracted as a set of movement primitives, can be thought of as a good initialization for a reinforcement learning (RL) agent. By capturing the essence of the shown demonstrations in as few parameters as possible, we simplify and increase the effectiveness of the skill transfer to the robot.

Sparsity is achieved in our framework in joint-space¹ by using a new iterative optimization approach, where a multi-task Elastic Net regression is alternated with a nonlinear optimization. The Elastic Net projects the solutions to a sparse set of features, and during the nonlinear optimization these features (the basis functions) are adapted to the data in a secondary optimization. Moreover these features are shared across multiple demonstrations, increasing the effectiveness of the feature learning strategy.

The fewer number of learned parameters using our iterative optimization procedure, compared to more traditional approaches, is independent of the robot DoF. This is a desirable property for Reinforcement Learning to adapt the learned parameters online. Moreover, by using

¹ Discarding the joint-level information and using only the Cartesian coordinates of the resulting movements, in a similar attempt to reduce the dimensionality of the robot learning problem, necessitates the use of inverse kinematics, running into feasibility and additional execution problems that might be artificially introduced.

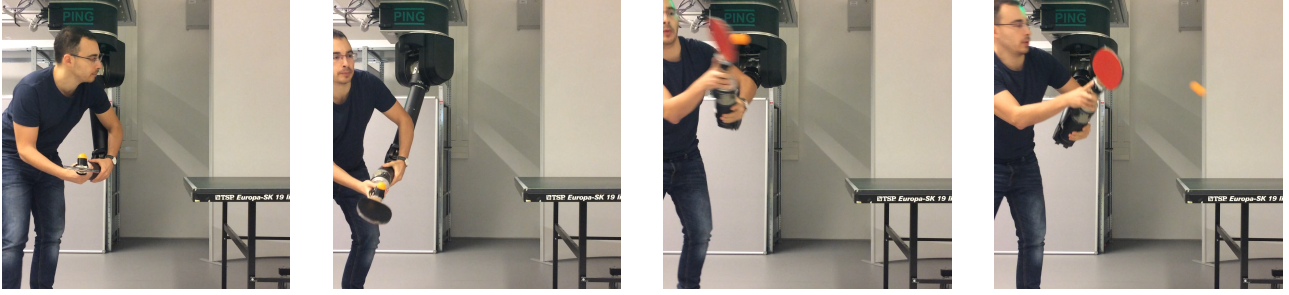


Figure 4.1: Our robot table tennis setup with a seven DoF Barrett WAM, where we demonstrate, using kinesthetic teach-in, multiple good table tennis serve movements while recording the resulting joint-space robot trajectories. A metal piece is attached to the end effector of the Barrett WAM, which connects to a standard sized table tennis racket. An egg-holder on the metal piece holds the ball initially before the serve. The demonstrator, after finding a good starting posture, starts by swinging the arm, giving the ball enough acceleration to propel it away from the robot. The ball is then hit in midair by a careful adjustment of the robot wrist. The initial posture, the swinging movement of the robot shoulder joints and the elbow, and finally the turning of the wrist all contribute to the style of the shown movement. Multiple demonstrations starting from different initial postures are recorded in one session. We compare and evaluate throughout the chapter different learning from demonstrations approaches using these demonstrations. We propose a new iterative optimization approach that can learn sparse parameters while adapting the features of the movement primitives to the demonstration data.

the Elastic Net path, we can rank the parameters in terms of importance, or effectiveness in explaining the demonstration data. We perform preliminary experiments on the Barrett WAM on a table tennis serve to validate the effectiveness of our new movement primitives.

In Chapter 2, a new trajectory generation framework in table tennis was introduced, where a free final-time optimal control problem was solved, generating minimum acceleration striking trajectories. This kinematic optimization approach was extended and evaluated in the real robot table tennis setup. The success of this and other similar model-based optimization approaches in dynamic tasks like table tennis heavily depends on the accuracy of the models. In the case of table tennis, an accurate *ball model* [67], [2] is especially difficult to acquire. The high spin rates make the ball flight difficult to model from first (physical) principles, while the various types of impacts make it also difficult to train machine learning approaches from raw ball position data. For the serve, an additional complication results from the ball take-off phenomena, which is similarly difficult to model or to learn.

Learning from demonstrations (LfD) is a promising framework for learning various robotic tasks efficiently without using hard-coded approaches or physical insights to model the specific aspects of each task. It has been used in many different robot scenarios to great effect, including robot manipulation and human-robot collaboration [68]. It was also useful in initializing the parameters of policy-search RL approaches for robot learning [46]. There are, by now, many different frameworks for LfD, including dynamical system representations such as the Dynamical Movement Primitives (DMP) [69], learning control Lyapunov-functions [70], and various other probabilistic approaches, such as the probabilistic movement primitives [71] or Gaussian mixture models [72]. These last two methods can, unlike DMPs, capture multiple demonstrations

in a parametric form, and can moreover be used to condition on way-points or different targets in joint or in task space. One particular disadvantage of all of the LfD approaches introduced above is that the features chosen to regress on the demonstration data are often manually tuned and the number of parameters to learn are explicitly specified. We think that fixing the features and tuning their hyperparameters for particular tasks harm the generalization and applicability of the movement primitives to novel scenarios.

The l_1 -regularized l_2 -norm regression (from hereon referred to as *Lasso*) is often used in the statistics and machine learning communities as a regression method that can simultaneously also perform automatic feature selection. A detailed introduction and analysis of Lasso can be found in [73]. Lasso was extended to the *multi-task* case (i.e., multi-output regression with shared features) in [74]. Our interest in Lasso lies in the fact that (multi-task) Lasso can perform systematic feature selection while training (multiple) movement primitives, augmenting the applicability of LfD to novel tasks. Moreover, selection and early pruning of features can be used to great effect in RL, possibly reducing the amount of interaction time with the real robot.

A new incremental procedure to solve ordinary least squares regression as well as Lasso problems was proposed in [4]. This algorithm, called *Least Angle Regression* or *LARS* for short, yields piecewise linear homotopy paths of the regression problem as a function of the l_1 -regularization term. These paths can be used to rank the features in terms of importance, as will be detailed later. Ranking the features of the trained movement primitives can reduce the curse of dimensionality in RL, decreasing as before the robot interaction time and possibly making the adapted movements also more interpretable to humans.

The *Elastic Net* imposing additional l_2 -regularization to Lasso was introduced in [15], where it was noted that a basic transformation converts the problem to a standard Lasso regression, and this is also valid in the multi-task setting. For the training of movement primitives, especially for dynamic trajectories like the table tennis serves, the *Elastic Net* with its l_2 -regularization can help to reduce the excessive accelerations throughout the learned movements, making them safer to implement on the robot.

In the next sections, we will detail how the sparse representation-learning of movement primitives can be formulated using the multi-task Elastic Net, coupled with nonlinear optimization on the feature parameters. To the best of our knowledge, the multi-task Elastic Net was not combined before with Radial Basis Functions in a (iterative) nonlinear feature selection and optimization framework. We also think that ranking the learned parameters in terms of importance is a new idea that can benefit the RL community.

4.2 Notation

The notation that we use throughout the chapter is standard: for a robot arm with n degrees of freedom (DoF), the joint configurations are $\mathbf{q} \in \mathbb{Q} = \{\mathbf{q} \in \mathbb{R}^n \mid \mathbf{q}_{\min} \leq \mathbf{q} \leq \mathbf{q}_{\max}\}$. The recorded joint positions over a movement is represented as a matrix $\mathbf{q}(t) \in \mathbb{R}^{N \times n}$ of N rows, with column $i = 1, \dots, n$ storing the positions throughout the movement corresponding to joint i .

Whenever multiple demonstrations are used for learning, i.e., $\mathbf{q}_{ij}(t)$ is recorded for $i = 1, \dots, n$ DoF and $j = 1, \dots, d$ demonstrations, these recordings are stacked to form the \mathbf{Q} matrix. The degrees of freedom are concatenated vertically in this case for a single demonstration, while the columns store the different demonstration data, i.e., $\mathbf{q}_{ij}(t) \rightarrow \mathbf{Q}_{N(i-1)+t/dt,j}$ for a recording of N time points with dt time intervals.

The Frobenius norm of a matrix is the square-root of the sum of its squared elements, $\|\mathbf{M}\|_F^2 = \sum_i \sum_j m_{ij}^2$, whereas the $\|\cdot\|_{21}$ norm used in the multi-task Elastic Net is defined instead as $\|\mathbf{M}\|_{21} = \sum_i \sqrt{\sum_j m_{ij}^2}$, i.e., l_2 -norm along the columns (degrees of freedom in our setting) and l_1 -norm along the rows (time steps). This norm is used to induce sparsity on the features, whose centers are initially located uniformly along the time axis.

4.3 Method

In this section, we discuss how one can acquire a sparse movement pattern from human demonstrations. We present first an algorithm that requires only a single human demonstration, and then present a suitable variant that can be employed for multiple demonstrations. This variant of the algorithm decouples the number of learned parameters from the degrees of freedom of the robot.

4.3.1 Learning a sparse representation from a single demonstration

Given a single demonstration $\mathbf{q}(\mathbf{t})$ at the (observed) time points \mathbf{t} , we'd like to extract a movement primitive that is sparse. That is, throughout the parametric optimization, we'd like to impose a good fit with as few basis functions as possible, while keeping the accelerations low during the trained movement pattern. Having low accelerations is beneficial both for robot safety as well as improving the tracking (execution) accuracy of the trajectories [2]. Mathematically, the criterion that we optimize can be written as

$$\min_{\boldsymbol{\beta}, \boldsymbol{\theta}} \|\mathbf{q}(\mathbf{t}) - \Psi(\mathbf{t}, \boldsymbol{\beta})\boldsymbol{\theta}\|_F^2 + \lambda_1 \|\boldsymbol{\theta}\|_{21} + \lambda_2 \|\ddot{\Psi}(\mathbf{t}, \boldsymbol{\beta})\boldsymbol{\theta}\|_F^2, \quad (4.1)$$

where $\Psi(\mathbf{t}, \boldsymbol{\beta}) \in \mathbb{R}^{N \times p}$ are the evaluations of the basis functions at \mathbf{t} , $\boldsymbol{\theta} \in \mathbb{R}^{p \times n}$ are the (sparse) regression parameters, and $\mathbf{q}(\mathbf{t})$ are the joint observations during the shown movement. The nonlinear radial basis functions (RBF) are parameterized by $\boldsymbol{\beta} \in \mathbb{R}^p$. An l_2 -penalty is put on the accelerations $\ddot{\Psi}(\mathbf{t}, \boldsymbol{\beta})\boldsymbol{\theta}$ of the extracted movement pattern $\Psi(\mathbf{t}, \boldsymbol{\beta})\boldsymbol{\theta}$, while a penalty with the l_1 -norm on the (rows of the) regression parameters $\boldsymbol{\theta}$ encourages sparsity of the found solutions.

This regression problem, for fixed $\boldsymbol{\beta}$, is known as the multi-task *Elastic Net* in the literature, where the features are shared among the sparse parameters along each degree of freedom. As opposed to the standard (multi-task) Lasso, the l_2 -norm penalty in the optimization (4.1) penalizing the accelerations throughout the motion, also adds stability to the Lasso solutions [15].

The solution to the weighted Elastic Net problem (4.1) for fixed $\boldsymbol{\beta}$ can be obtained by transforming the problem to an equivalent (unweighted) Lasso problem, solving it via a convex optimizer (e.g., *coordinate descent* is very effective for Lasso problems), and then transforming the solutions back to the Elastic Net parameters.

We can solve the original problem (4.1) iteratively (as in Expectation-Maximization type of algorithms) by first starting the iteration with a Lasso solution of an overly-parameterized radial basis function regression. At each iteration, the RBF parameters $\boldsymbol{\beta}_i$ corresponding to the basis functions with nonzero Lasso regression parameters $\boldsymbol{\theta}_{ij} > 0$, $j = 1, \dots, n$ are updated for each $i = 1, \dots, p$ via nonlinear optimization. The Elastic Net regression is then performed, and the features corresponding to parameters with zero coefficients are removed. These two alternating

steps can be continued till convergence, or rather terminated in a fixed number of steps. The iterations converge when the change in function value of the total cost in (4.1) is below a certain tolerance ϵ . Depending on the initial solution parameters β_0 and θ_0 , the iteration converges to a local minimum.

The full procedure is shown in Algorithm 5 in detail. We call the resulting algorithm *Learning Sparse Demonstration Parameters* or *LSDP* for short. The algorithm alternates between the multi-task Elastic Net (lines 4 and 10) and the nonlinear optimizer (BFGS, in line 8). In between, the zero entries of the regression parameters θ and the corresponding columns of $\Psi, \dot{\Psi}$ are removed in the Prune step (lines 5 and 13). The pruning operation simplifies the optimization in the upcoming iterations, as the removed RBF parameters cannot then be re-elected later. We use the squared exponential kernel to construct our basis functions, i.e., for every i, j we use

$$\Psi_{ij}(t_i) = \exp(-(t_i - \mu_j)^2 / (2\sigma_j^2)), \quad (4.2)$$

to form the (i, j) 'th element of the matrix Ψ . The data is initially centered (line 2), i.e., the mean of each joint recording is subtracted from the signal, and the means \mathbf{q}_0 are stored as the intercepts for the particular demonstration.

For a good performance of the algorithm, i.e., obtaining low residuals with a sparse set and low accelerations, choosing the regularizer weights λ_1 and λ_2 suitably is crucial. These parameters can be set using cross-validation either before Algorithm 5 or together with the initial regression (line 4). The regularizers should be scaled down accordingly with the decreasing residual norms (see line 12), otherwise the algorithm can converge to the empty set for the parameters θ .

The optimization problem, depending on the parameterization and the features used, can be highly nonconvex, possibly with many local minima. The number of local minima, fortunately, does not seem to pose a problem in terms of residual norm. As long as the initial representation is sufficiently (over) parameterized, most solutions fit well to the demonstration data. For more sparse representations, however, one may choose to restart the training procedure a few times from perturbed initial conditions, especially for the RBF parameters β . See the Experiments section for more discussion on the implementation details.

The computational complexity of the algorithm overall is dominated by the complexity of the multi-task Elastic Net step (line 10), where the coordinate descent algorithm is used to solve a Lasso problem (after a transformation in constant time $\mathcal{O}(1)$). The time-complexity of the LARS algorithm to solve Lasso problems is known to be $\mathcal{O}(Np^2)$ [4], but coordinate-descent converges often faster, in our experience. One step of Quasi-Newton methods has time-complexity $\mathcal{O}(p^2)$ (plus the cost for function and gradient evaluations [3]), coming from the matrix multiplication operations. Quasi-Newton optimization may, depending on the initialization, require many of these steps, in our case we limit it to 1000 steps for each iteration of *LSDP*.

4.3.2 Coupling the parameters across dimensions

The algorithm *LSDP* discussed in the previous subsection uses the multi-task Elastic Net to enforce the same basis functions for each degree of freedom (along the columns of $\mathbf{q}(t)$ and the parameter matrix θ), while the parameter vectors corresponding to each joint movement are different and optimized independently: the regression parameters are decoupled across the degrees of freedom (DoF) of the robot. In particular, the number of regression parameters grow

Algorithm 5 Learning sparse parameters with regression (*LSDP*) for a single demonstration

Require: $\mathbf{q}, \mathbf{t}, \boldsymbol{\mu}, \boldsymbol{\sigma}^2, \lambda_1, \lambda_2, \epsilon > 0$

- 1: Initialize $\boldsymbol{\beta}_0 = [\boldsymbol{\mu}, \boldsymbol{\sigma}^2]$
- 2: Center the data, $\mathbf{q}_0, \mathbf{q} \leftarrow \text{Center}(\mathbf{q})$
- 3: Form $\Psi, \ddot{\Psi}$ using $\boldsymbol{\beta}_0$ and \mathbf{t}
- 4: $\boldsymbol{\theta}_0 \leftarrow \text{MultiTaskElasticNet}(\Psi, \ddot{\Psi}, \mathbf{q}, \lambda_1, \lambda_2)$
- 5: $\boldsymbol{\theta}_0, \boldsymbol{\beta}_0 \leftarrow \text{Prune}(\boldsymbol{\theta}_0, \boldsymbol{\beta}_0)$
- 6: Form $\Psi, \ddot{\Psi}$ using $\boldsymbol{\beta}$ and \mathbf{t}
- 7: **repeat** $k = 1, \dots,$
- 8: $\boldsymbol{\beta}_k \leftarrow \text{BFGS}(\Psi, \ddot{\Psi}, \boldsymbol{\beta}_{k-1}, \boldsymbol{\theta}_{k-1}, \mathbf{q}, \lambda_1, \lambda_2)$
- 9: Form $\Psi, \ddot{\Psi}$ using $\boldsymbol{\beta}_k$ and \mathbf{t}
- 10: $\boldsymbol{\theta}_k \leftarrow \text{MultiTaskElasticNet}(\Psi, \ddot{\Psi}, \mathbf{q}, \lambda_1, \lambda_2)$
- 11: Calculate residual norm r_k , total cost f_k using (4.1)
- 12: Scale penalties $\lambda_i \leftarrow \lambda_i r_k^2 / r_{k-1}^2, i = 1, 2$
- 13: $\boldsymbol{\theta}_k, \boldsymbol{\beta}_k \leftarrow \text{Prune}(\boldsymbol{\theta}_k, \boldsymbol{\beta}_k)$
- 14: Form $\Psi, \ddot{\Psi}$ using $\boldsymbol{\beta}_k$ and \mathbf{t}
- 15: **until** $\|f_k - f_{k-1}\| < \epsilon$

linearly with the robot DoF, which may be undesirable for applying policy search RL approaches to high dimensional robotic systems especially.

Furthermore, the algorithm has to be applied for each demonstration separately, i.e., there is no *coupling* or information shared between the demonstrations. In order to enforce rather the features to be shared *across demonstrations* rather than the robot DoFs, we discuss here a variant of the algorithm *LSDP*, which we call coupled *LSDP*, or *cLSDP* for short.

The algorithm *cLSDP*, shown in Algorithm 6, requires only a few changes compared to Algorithm 5. The data is centered for each demonstration to obtain the intercepts \mathbf{Q}_0 . The algorithm then stacks (lines 1–3) the dependent regression variables \mathbf{q}_i and the RBF parameters $\boldsymbol{\beta}_i$ vertically for each degree of freedom $i = 1, \dots, n$ to form the matrices $\mathbf{Q} \in \mathbb{R}^{Nn \times d}$ and $\Psi \in \mathbb{R}^{Nn \times p}$. The second time derivative of the data matrix, $\ddot{\Psi}$, is stacked as well to form the regression model as in (4.1).

As opposed to *LSDP*, in this procedure there are n times the number of RBF parameters $\boldsymbol{\beta}$ to be optimized (line 8), as the features are adapted independently for each DoF. The regression parameters $\boldsymbol{\theta}$, on the other hand, are coupled across the DoFs, and their cardinality is reduced by n times. The nonlinear optimization computational complexity in this case dominates that of the multi-task Elastic Net and the net result is roughly a n times increase in the computation time between each iteration of *cLSDP*.

Note that the parameters for each demonstration are estimated together, i.e., the columns of the $\boldsymbol{\theta}$ matrix correspond to the regression parameters for different demonstrations. One way to generalize the learned movement primitives to different task conditions (such as varying initial joint states) would be to interpolate between these regression parameters. A policy could then be effectively created, whose generalization would be limited by the number and the quality (e.g. variety, success rate) of the demonstrations.

Algorithm 6 Learning coupled sparse parameters with regression (*cLSDP*) across multiple demonstrations

Require: $\mathbf{q}_{ij}, \mathbf{t}, \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2, \lambda_1, \lambda_2, \epsilon > 0$

- 1: Stack \mathbf{q}_{ij} to form \mathbf{Q} , $i \in [1, n], j \in [1, d]$
- 2: Center the data, $\mathbf{Q}_0, \mathbf{Q} \leftarrow \text{CenterStacked}(\mathbf{Q})$
- 3: Stack $\boldsymbol{\beta}_0 = [\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_n, \boldsymbol{\sigma}_1^2, \dots, \boldsymbol{\sigma}_n^2]$
- 4: Stack $\boldsymbol{\Psi}, \ddot{\boldsymbol{\Psi}}$ using $\boldsymbol{\beta}_0$ and \mathbf{t} across DoFs
- 5: $\boldsymbol{\theta}_0 \leftarrow \text{MultiTaskElasticNet}(\boldsymbol{\Psi}, \ddot{\boldsymbol{\Psi}}, \mathbf{Q}, \lambda_1, \lambda_2)$
- 6: $\boldsymbol{\theta}_0, \boldsymbol{\beta}_0 \leftarrow \text{PruneStacked}(\boldsymbol{\theta}_0, \boldsymbol{\beta}_0)$
- 7: Stack $\boldsymbol{\Psi}, \ddot{\boldsymbol{\Psi}}$ using $\boldsymbol{\beta}$ and \mathbf{t} across DoFs
- 8: **repeat** $k = 1, \dots,$
- 9: $\boldsymbol{\beta}_k \leftarrow \text{BFGS}(\boldsymbol{\Psi}, \ddot{\boldsymbol{\Psi}}, \boldsymbol{\beta}_{k-1}, \boldsymbol{\theta}_{k-1}, \mathbf{Q}, \lambda_1, \lambda_2)$
- 10: Stack $\boldsymbol{\Psi}, \ddot{\boldsymbol{\Psi}}$ using $\boldsymbol{\beta}_k$ and \mathbf{t} across DoFs
- 11: $\boldsymbol{\theta}_k \leftarrow \text{MultiTaskElasticNet}(\boldsymbol{\Psi}, \ddot{\boldsymbol{\Psi}}, \mathbf{Q}, \lambda_1, \lambda_2)$
- 12: Calculate residual norm r_k , total cost f_k using (4.1)
- 13: Scale penalties $\lambda_i \leftarrow \lambda_i r_k^2 / r_{k-1}^2$, $i = 1, 2$
- 14: $\boldsymbol{\theta}_k, \boldsymbol{\beta}_k \leftarrow \text{PruneStacked}(\boldsymbol{\theta}_k, \boldsymbol{\beta}_k)$
- 15: Stack $\boldsymbol{\Psi}, \ddot{\boldsymbol{\Psi}}$ using $\boldsymbol{\beta}_k$ and \mathbf{t} across DoFs
- 16: **until** $\|f_k - f_{k-1}\| < \epsilon$

4.3.3 Ranking the demonstration parameters

The regression parameters estimated with *cLSDP* can also be ranked in terms of statistical significance, i.e., correlation. The Elastic Net *regularization path* of the *LARS* algorithm [4] traces the evolution of the parameters as the l_1 -penalty weight λ_1 of equation (4.1) increases. An example regularization path for only eight selected regression parameters $\boldsymbol{\theta} \in \mathbb{R}^8$ are plotted in Figure 4.2. Initially when the regularization is low ($\lambda_1 \approx 0$) on the right side of the Figure, the coefficients are close to their (nonzero) values in ordinary Least Squares. As the regularization term increases, some of these terms drop out, i.e., the coefficients become zero as the path is traced towards the left-hand side of the Figure. The corresponding features can then be eliminated from the regression model, leading not only to a sparse, but also a ranked set of features.

In the proposed method *cLSDP*, the *LARS* algorithm instead of coordinate descent can be used in the final Elastic Net computation step (line 11 of Algorithm 6) to generate the full regularization path. The addition of the selected movement primitive parameters can then be traced. An example path for twenty parameters selected by the Algorithm is plotted in Figure 4.2. These parameters can be ranked according to their evolution, i.e., the coefficients that early on during the path become nonzero are likely to signal more causally effective components of the motion. For example, in the shown plot, the parameters corresponding to the red lines, would be ranked after some of the parameters appearing before (black lines). More prominent components of the motion can be identified this way. These movement components could be adapted earlier with RL strategies, reducing the curse of dimensionality in high dimensional robot learning problems.

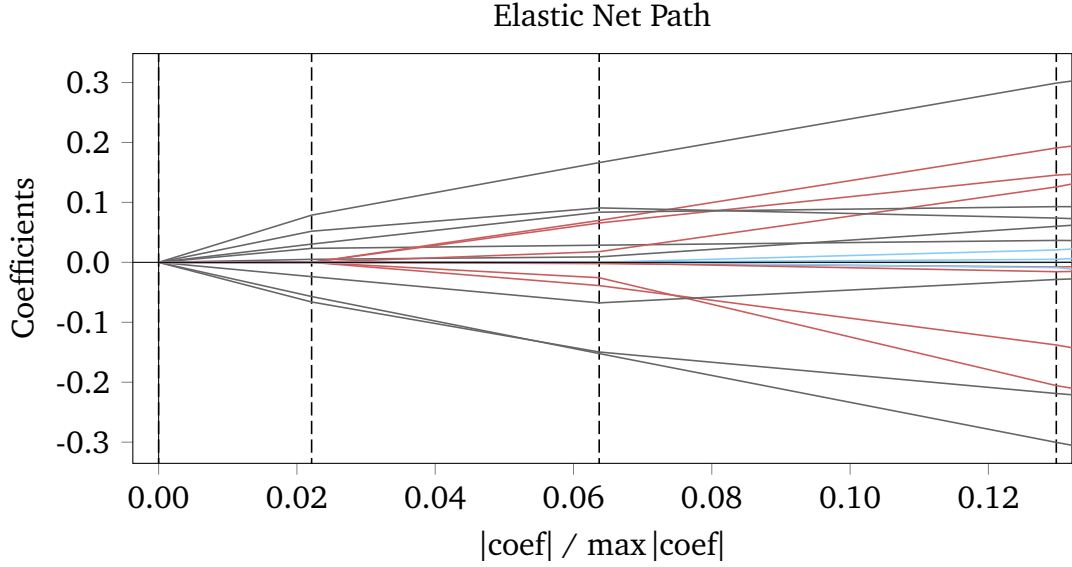


Figure 4.2: An example Elastic Net path with twenty selected parameters is shown after training Algorithm 6, *cLSDP*, with five demonstrations. This *regularization path* can be generated in the final step of the algorithm. As the l_1 -penalty term λ_1 of the regression problem (4.1) is reduced, the coefficients converge to their (maximal) ordinary least squares values at the right hand side of the plot (not shown). Each dashed line signals a change in the regularization term, and the coefficients are updated accordingly. The algorithm *LARS* [4] can be used to generate these piece-wise linear regularization paths. One possible way to use this path is to rank the sparse parameters of the learned movement primitives in terms of statistical importance. For example, in the shown plot, the parameters corresponding to the red lines would be ranked after the other parameters appearing before (black lines). The parameter paths, whose coefficients become nonzero close to each other, are drawn with the same color.

4.4 Experiments

In this section, we conduct experiments to learn a sparse set of movement primitive parameters using the proposed approaches (see Algorithms 5 and 6). The two algorithms are also compared against two competing movement primitive learning methods (DMPs and l_2 -regularized regression). Finally we present real robot experiments on our table tennis platform where we show that the learned sparse movements nevertheless look similar to the shown demonstrations in style. They can also be implemented safely on the robot.

4.4.1 Learning from Demonstrations

The algorithms *LSDP* and its coupled variant *cLSDP*, discussed in Section 4.3, are applied here on the demonstrated Barrett WAM serve movements, see Figure 4.1. From a continuous stream of joint values, recorded at 500 Hz during a kinesthetic teach-in session, a predetermined number of d movements are selected by detecting the maximum d velocities in joint space and windowing around these points for a fixed duration of one second. We implement the preprocessing

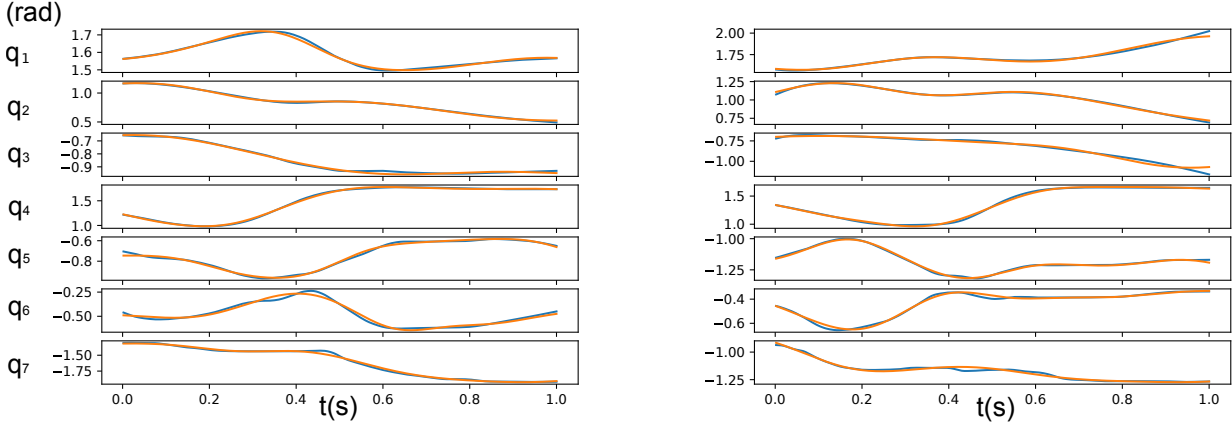


Figure 4.3: Two movement primitives learned by the proposed algorithm *cLSDP*, are plotted in joint space against the recorded demonstrations. The table tennis serve movements, shown in blue, after preprocessing and segmenting the recorded time series are one second long each. The first three rows, q_1 through q_3 , correspond to the shoulder movement in joint space. The fourth row q_4 shows the movement of the elbow. Finally, the last three rows (q_5 through q_7) show the wrist movements in joint space. The trained movement primitives, shown in orange, couple the sparse regression parameters across the degrees of freedom of the robot.

as well as the Algorithms in Python, using the *scikit-learn* toolbox for the multi-task Elastic Net and the *scipy* toolbox for the nonlinear optimization (BFGS, see lines 8 and 9 in the Algorithms, respectively).

The preprocessed examples using the above procedure result in the joint matrix $\mathbf{q}(t) \in \mathbb{R}^{500 \times 7}$ for each example demonstration. For the algorithm *LSDP*, the initial RBF centers $\boldsymbol{\mu}_0 \in \mathbb{R}^{500}$ are placed at every time point and the RBF widths σ_0^2 are set uniformly to 0.1. The algorithm stretches, prunes and expands the basis functions throughout the optimization to produce a very sparse, nonuniform set of basis functions shared across the seven degrees of freedom (DoF). The columns of the regression parameter matrix $\boldsymbol{\theta}$, on the other hand, are separate for each DoF.

The Algorithm *cLSDP*, on the other hand, optimizes n times more RBF parameters, i.e., $\boldsymbol{\mu} \in \mathbb{R}^{3500}$ and $\boldsymbol{\sigma}^2 \in \mathbb{R}^{3500}$ for the Barrett WAM with $n = 7$. During the optimization, all of the recorded data from d demonstrations are used together, and the same set of basis function parameters $\boldsymbol{\beta} = [\boldsymbol{\mu}^T, (\boldsymbol{\sigma}^2)^T]^T$ are learned across multiple demonstrations. The learned parameters $\boldsymbol{\mu}, \boldsymbol{\sigma}^2, \boldsymbol{\theta}$, along with the intercepts, are saved after the optimizations to a json file, to be loaded later by the real-time robot controller in C++ during the online experiments.

Table 4.1 summarizes the results of learning movement primitives from five different demonstrations. The three columns used to compare the different approaches show on average the number of features selected (equivalently, the number of regression parameters with nonzero coefficients), the norm of the second derivatives of the trained movement primitives and the norm of the residuals, respectively. The five demonstration parameters are estimated together in *cLSDP*, whereas *LSDP* is run separately for each demonstration to obtain the mean and the standard deviations reported in the table. Note that the number of parameters in total used by *cLSDP* (37) is much lower than the on-average 16.8 parameters used by *LSDP* for each

robot DoF. The residual is slightly higher, this is a result of the parameters being shared across the dimensions. In particular, we have observed that *cLSDP* does not fit the last three joints, corresponding to the Barrett WAM wrist, as tightly as *LSDP*. This could be because the motion of the wrist is highly varying across the movements and the coupling of the features induced by the algorithm across demonstrations brings these movements closer.

The two proposed algorithms are compared against two baselines in Table 4.1. The first baseline is the Dynamic Movement Primitives (DMPs) with a fixed number of basis functions. DMPs learn the parameters of an attractor dynamics, i.e., a set of differential equations that converge to a suitably chosen goal state [69]. A standard regression is performed on the estimated attractor dynamics accelerations. The second baseline is l_2 -penalized standard regression, with the penalty on the accelerations. During the experiments we used a total of ten basis functions both for the DMPs and for the l_2 -penalized regression. The basis functions are spread uniformly, as discussed before for the proposed algorithms, around the one second long (preprocessed) demonstrations.

DMPs, as a result of the dynamic constraint of reaching a desired goal position, can incur very high initial accelerations in joint space. Even if the hyperparameters are optimized accordingly to prevent such high accelerations, slight modifications of initial joint positions can again give rise to high accelerations. The suggestion proposed in [75] to modify the accelerations with the phase can reduce the initial accelerations, but then we have found that the convergence to the goal positions can suffer drastically. The fixed basis function regression does not have this problem, but as in DMPs, optimizes a fixed number of parameters. As shown in Table 4.1, the number of parameters to fit the demonstrations well is, for both compared methods, on average double the number optimized by *cLSDP*.

See Figure 4.3 for two example regression results. The demonstrated movements are shown in blue and the regression results are shown in orange. The first three rows, q_1 through q_3 , correspond to the shoulder movement in joint space. The fourth row q_4 shows the movement of the elbow. Finally, the last three rows (q_5 through q_7) show the wrist movements in joint space, see Figure 4.1 for the way the human can hold the robot and show the demonstrations.

Three example demonstrations are plotted in task space in Figure 4.4 along with the recorded ball positions, detected and triangulated from two cameras opposite to the robot. The initial positions of the racket center and the ball in the egg-holder are marked as 0 in red and blue, respectively. The egg-holder is at a distance of roughly 14 cm to the racket center. During the movement the ball is hit by the human demonstrator moving the robot arm, and as the demonstrator slows down the motion to a halt, the ball is seen flying towards the table.

Table 4.1: Comparison of different learning from demonstrations approaches, averaged over five different serve demonstrations

	No. parameters ($\ \theta\ _0$)	Acceleration norm	Residual norm
<i>LSDP</i>	$(16.8 \pm 3.25) \times 7$	59.04 ± 7.0	0.59 ± 0.11
<i>cLSDP</i>	37	55.98 ± 11.78	0.73 ± 0.09
<i>DMPs</i>	11×7	621.73 ± 57.45	0.92 ± 0.06
<i>l2-reg. regr.</i>	11×7	215.45 ± 35.25	2.12 ± 0.47

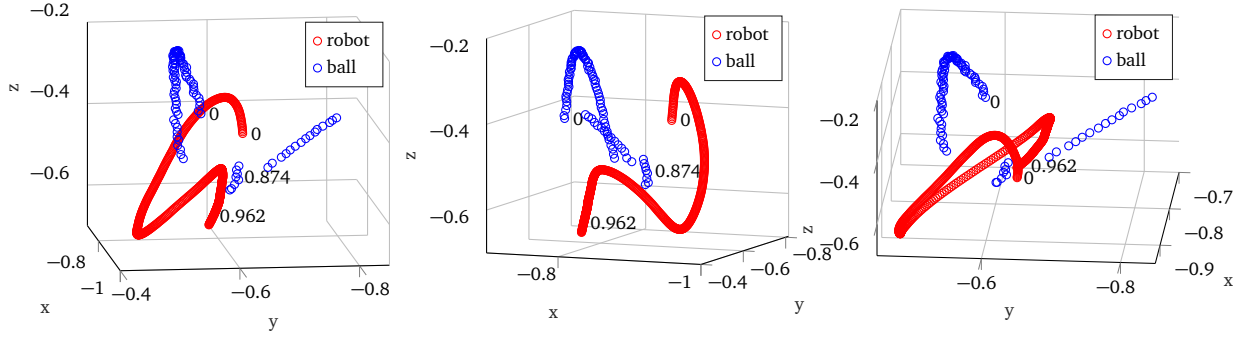


Figure 4.4: Three example demonstrations in task space. The initial position of the racket center and the ball in the egg-holder are marked as 0 in red and blue circles, respectively. The egg-holder is located approximately 14 cm away from the racket centre. Before the racket stops moving, the ball is already hit, flying towards the table.

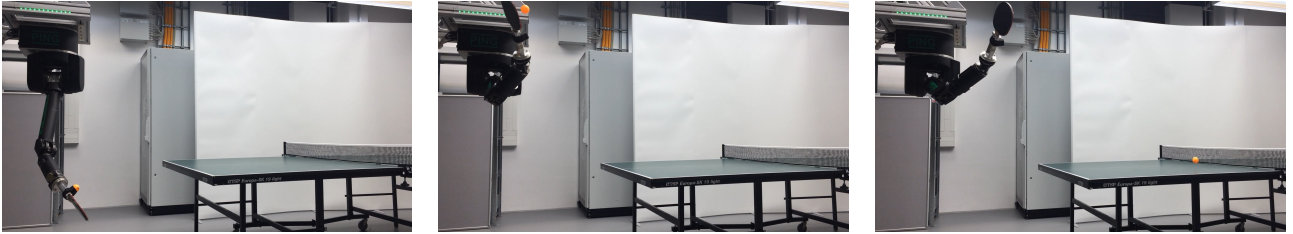


Figure 4.5: A successful rollout during real robot experiments. The ball is initially on top of the egg-holder and during the movement, as a result of the acceleration of the arm, it takes-off from the robot, to be later hit by the racket towards the table. The arm then decelerates towards a safe resting posture.

4.4.2 Robot Experiments

Finally, we conduct experiments in our real robot table tennis platform, see Figure 4.1. Our table tennis playing robot is a seven degree of freedom Barrett WAM arm that is capable of reaching high accelerations and velocities. However it is cable-driven and high accelerations can cause the cables to break easily. A standard size racket is attached to the end-effector via a metal bar. The racket has a radius of roughly $r_R = 7.6$ cm. The table and the table tennis balls are standard sized, balls have a radius of 2 cm, and the table geometry is roughly $276 \times 152 \times 76$ cm. Throughout the experiments, the Barrett WAM is placed at a distance of about one meter to the end of the table and its base is located 95 cm above the table. This makes it difficult (but not impossible) for the robot to hit the table. An egg-holder holds the table tennis ball initially, wrapped around the metal bar connecting the end-effector and the racket, see Figure 4.1.

A successful serve in our robot platform is shown in Figure 4.5. The ball is initially placed on top of the egg-holder (approximately 14 cm away from the racket center along the racket plane). The movements captured by the algorithm *cLSDP* are then executed on the robot. During the movements, as a result of the robot's accelerating motion, the ball takes off from the robot arm. The ball is then hit by the robot towards the table. The arm finally decelerates towards a resting posture as the ball lands on the robot court, passes the net, and lands again on the opposite side. We notice that the initial accelerating motion and the final wrist movement are critical for a good serve. Without the initial accelerations, the ball has no chance to take-off, and without

the final wrist movement (e.g., a quick rotation towards the ball) the ball is not hit well towards the table.

A video showing some demonstrated movements, as well as several actual rollouts on the Barrett WAM is available online: https://youtu.be/vj6jfx_MQmQ. Note that orchestrating the right movement during the demonstrations can be quite difficult, as moving the shoulder and the elbow joints can feel very awkward depending on the posture. When a good initial posture (both for the robot and the demonstrator) is found, the resulting demonstrations have a higher quality in general. These higher quality demonstrations also have a higher chance of being executed successfully.

Comparing our approach to the DMPs, we notice that the DMPs immediately start the movement with very large accelerations, these can be dangerous for the robot and the low-level Barrett WAM controllers do not support 80% of the movements. DMPs are good at capturing movements that converge to goal positions (with zero or low velocities), however they are less accurate in capturing the style (e.g., initial movement, final wrist turns) of dynamics movements such as table tennis serves, without manual tuning (the number of basis functions, locations and widths of the basis functions, etc.) for each task. We have seen that *cLSDP*² on the contrary, can capture the style of the movement, as shown in Figure 3 for some of the movements, with a sparse set of basis functions. The generalization capacity of these selected basis functions hinges on the quality and the number of the shown demonstrations.³

4.5 Conclusion

In this chapter we presented a new learning from demonstrations (LfD) approach to represent and learn table tennis serve movements. The proposed algorithms *LSDP* and *cLSDP* learn sparse parameters of the radial basis functions (RBF) from single and multiple demonstrations, respectively. The algorithms employ iterative optimization, alternating between a weighted multi-task Elastic Net regression step that learns sparse parameters given the features and a nonlinear optimization step that adapts the features (more specifically, the widths and centers of the RBFs corresponding to the nonzero regression parameters). The algorithm *cLSDP*, unlike *LSDP*, learns (sparse) parameters that are independent of the robot DoF. This desirable property is achieved by having different basis functions that are adapted across each DoF separately. The multi-task Elastic Net, in this case, forces the joint-dependent features to be shared across multiple demonstrations.

The cost function chosen for the optimization includes the residual of the fit, as well as l_2 -regularization terms on the accelerations and l_1 -regularization on the regression coefficients. We compared the performance of the proposed algorithms with Dynamic Movement Primitives (DMPs) and the standard l_2 -regularized regression, and we evaluated the performance of each on the different components of the chosen cost function (see Table 4.1). Finally, we discussed the performance of the actual rollouts, using our framework, on the real robot table tennis

² Note that executing the Algorithm *LSDP*, trained on each demonstration independently, shows a very similar performance, to that of Algorithm *cLSDP* at the moment. However we expect improvements on the learning performance, if Reinforcement Learning is applied on top of the more sparse set of *cLSDP* parameters.

³ If the number and the quality of the demonstrations is not enough, then the selected features and their ranking (using the regularization path) may be spurious, i.e., without any meaningful physical relevance. Increasing the number and the quality (e.g., increased variety of movements, higher success rates) of the movements could remedy such a limitation.

setup. One can see in the video available online that the style of the movements are preserved while maintaining low accelerations throughout the motion, which is important for the safety of the robot.

The sparsity of the parameters, as well as their decoupling from the robot DoF, is a desirable property for policy-search RL approaches that could adapt the regression parameters online based on a suitable reward function. We have presented a way to rank these policy parameters, in the last subsection of Section 4.3, based on how well the parameters explain the (multiple) demonstration recordings. We think that this is a promising research direction to combat the curse of dimensionality in high dimensional robot learning tasks, and we will focus on it more in future experiments.

5 Conclusion

To conclude, we will first start by summarizing the contributions and the content of the thesis. We will then mention some of the open problems that are important research topics. Aside from extending the work presented here in relevant directions, some of these points are *crucial* for increasing robot table tennis performance to human-like levels. Note that for much of the work, optimization algorithms or numerical routines from optimal control were crucial. We think that further progress in dynamic robotics tasks will most likely come from the extension and tight integration of various optimization routines in learning and control.

5.1 Summary

In this thesis we have presented and analyzed new algorithms for trajectory generation and learning control. We started by introducing in Chapter 2, an optimal trajectory generation framework for robot table tennis. The resulting optimizations are based on feedforward solutions, given by the Minimum Principle. To make our optimization framework practical, we identified the parameters of spinning ball models from offline data. We could then estimate the incoming ball state and a spin component online from camera observations. The estimated ball states are used online to predict the incoming ball trajectory, and also to plan for the post-strike ball trajectory. We presented two different algorithms that lead to two different table tennis players, and evaluated their performance in our robot table tennis setup.

For our table tennis experiments, we used a high-gain PD controller to track the computed trajectories aggressively. This leads to accurate tracking only for slow or short trajectories, For longer or faster desired trajectories, the performance suffers as the deviations from the reference grow over time. Tracking the desired racket velocities are critical for a successful performance. We therefore consider in Chapter 3 accurate trajectory tracking, as a prerequisite for a good table tennis performance. We develop a new adaptive, cautious and recursive Iterative Learning Control update law, that is based on Bayesian model adaptation. The linear time varying model matrices (describing the propagation of the errors around the reference) are updated at each iteration with Linear Bayesian Regression. The variances are then used in the (closed-form) cautious ILC update as effective regularization terms. We show the performance of our approach in extensive simulations and in real robot experiments. Moreover, we discuss in detail the effects of adding a forgetting factor in the adaptation. A more biased procedure that updates only the rigid body link parameters is also mentioned.

In the last chapter, we considered a learning from demonstrations framework for a robot table tennis serve task. The acquired human demonstrations from the kinesthetic teach-in (with recorded joint positions) are used to learn a sparse set of movement primitive parameters. The movements are parameterized by both radial basis function parameters as well as the regression parameters. A nonlinear optimization problem that penalizes the l_1 -norm of the regression coefficients as well as the accelerations of the resulting movement is solved iteratively. The resulting regression parameters, using the Multi-Task Elastic Net, is sparse, independent of the dimensionality of the robot, and can be ranked in terms of importance. The resulting learning

pipeline leads to a very compact representation for movement primitives, which can be exploited by reinforcement learning approaches.

5.2 Open Problems

Throughout the thesis, we have mentioned in each chapter some of the open problems that might be addressed in a future work. Here in this subsection, we come back to the open problems with a renewed holistic understanding, keeping in mind the challenges of the table tennis task.

5.2.1 Higher Level Strategy

The two table tennis algorithms presented in Chapter 2 are evaluated in our robot table tennis setup. The table tennis experiments are run online with a ballgun facing the robot. The two algorithms, or players, lead to two different play styles. The Focused Player focuses on returning the ball to a fixed position on the opponent's court, while the Defensive Player considers the whole table (as a feasible region.)

It would be interesting to see a higher level strategy, that based on a suitable game state (e.g., position, style and game points of the opponent) could switch (as well as parameterize) between the two algorithms as needed. A less hierarchical approach would be to have the optimization change freely based on the game state, e.g., by incorporating an extra parameter that transitions smoothly between different styles of play. Incorporating variance in the decision making process could partly answer this point. We discuss this point in the next subsection.

5.2.2 Incorporating Uncertainty in Trajectory Generation

We have presented the details of the ball estimation in the Experiments section of Chapter 3. Whenever there is a table tennis ball approaching the robot, the means and variances of the incoming ball state is updated with an (Extended) Kalman Filter after each observation. The means are used to predict the incoming ball path around the workspace of the robot, while the variances are ignored in the decision making (except for outlier detection). A more *cautious* trajectory generation algorithm would include the variances within the optimization to generate trajectories more robustly. Depending on the formulation of the optimization criteria, an incoming ball with large variance could, for example, produce a slow trajectory that may not intersect exactly with the mean of the predicted ball positions. Obtaining more reliable observations decrease the variance and the robot could then update the old trajectory with a new one that approaches the incoming mean ball positions more aggressively. We believe that this could be a way to improve the performance of the robot online, as well as a means to automate the (so far mostly manually chosen) trajectory initiation and correction process. Ball and robot model uncertainties can also be incorporated in such a framework.

The Stochastic Minimum Principle literature might help in coming up with a feasible optimization framework. The resulting optimization would in this case be naturally run online repeatedly as in Model Predictive Control (MPC). As an alternative, computing local feedback policies (as a function of also the variances, rather than just the means) offline could replace MPC.

In this framework, the desired ball landing positions could as well be a random distribution over the table, and the cost could include expectations of the ball landing positions or a divergence term (e.g., Kullback-Leibler divergence). An opponent giving a strong, not-so-well-observed spin would result in large variance in after-strike ball landing positions, possibly pushing the robot to a more defensive play, while a simpler incoming ball (e.g., slower or without much spin) would be more amenable to focused, or more aggressive, hitting behavior.

5.2.3 Generalizing with Iterative Learning Control

Although we have presented and analyzed a new efficient model based Iterative Learning Control (ILC) algorithm, we had to evaluate its performance independently for each new trajectory. The models that we learned this way were adapted from the same nominal model (with the same set of link parameters) each time that we ran our experiments on the Barrett WAM. This is mostly due to the fact that ILC is a local method, i.e., the feedforward controls and the feedback matrices learned between iterations are only *locally* valid only around a particular reference trajectory.

Generalizing between performances is in general an open problem for ILC methods. One straightforward approach is to interpolate between the observed trajectories, e.g., with a Gaussian Process. This can be done in several different ways, but a simple approach could be to fit a forward dynamics model. While the simpler linear time varying models of the proposed ILC is updated online, an offline batch procedure, could slowly update a forward dynamics model, which would offer better generalization guarantees than various other forms of interpolation between trajectories.

If the forward dynamics model (e.g., rigid body dynamics) is believed to be of a parametric form, a simple way to generalize would also be to simply adapt the link parameters. A semi-parametric model could try to achieve the best of both worlds, i.e., first adapt the parametric model and then fit a nonparametric model on the residuals. We considered some of these options in Chapter 2, but we did not evaluate their effectiveness in generalizing across different trajectories.



Bibliography

- [1] O. Koc, G. Maeda, and J. Peters, “A new trajectory generation framework in robotic table tennis,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.
- [2] O. Koc, G. Maeda, and J. Peters, “Online optimal trajectory generation for robot table tennis,” *Robotics and Autonomous Systems*, vol. 105, pp. 121 – 137, 2018.
- [3] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer-Verlag, 1999.
- [4] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, “Least angle regression,” *Annals of Statistics*, vol. 32, pp. 407–499, 2004.
- [5] R. L. Anderson, *A Robot Ping-pong Player: Experiment in Real-time Intelligent Control*. Cambridge, MA, USA: MIT Press, 1988.
- [6] J. Billingsley, “Robot ping pong, practical computing,” 1983.
- [7] H. Fässler, H. A. Beyer, and J. Wen, “A robot ping pong player: optimized mechanics, high performance 3d vision, and intelligent sensor control,” *Robotersysteme*, vol. 6, no. 3, pp. 161–170, 1990.
- [8] M. Ramanantsoa and A. Durey, “Towards a stroke construction model,” *Int. Journal of Table Tennis Science*, vol. 2, pp. 97–114, 1994.
- [9] K. Mülling, J. Kober, and J. Peters, “A biomimetic approach to robot table tennis,” *Adaptive Behavior*, vol. 19, no. 5, pp. 359–376, 2011.
- [10] H. Li, H. Wu, L. Lou, K. Kühnlenz, and O. Ravn, “Ping-pong robotics with high-speed vision system,” in *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, pp. 106–111, Dec 2012.
- [11] Y. Huang, D. Xu, M. Tan, and H. Su, “Adding active learning to lwr for ping-pong playing robot,” *IEEE Transactions on Control Systems Technology*, vol. 21, pp. 1489–1494, July 2013.
- [12] K. Mülling, J. Kober, O. Kroemer, and J. Peters, “Learning to select and generalize striking movements in robot table tennis,” *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.
- [13] Y. Huang, D. Büchler, O. Koç, B. Schölkopf, and J. Peters, “Jointly learning trajectory generation and hitting point prediction in robot table tennis,” in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 650–655, Nov 2016.
- [14] D. A. Kendrick, *Stochastic Control for Economic Models*. 2002.
- [15] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the Royal Statistical Society, Series B*, vol. 67, pp. 301–320, 2005.

-
- [16] M. Matsushima, T. Hashimoto, M. Takeuchi, and F. Miyazaki, "A learning approach to robotic table tennis," *IEEE Transactions on Robotics*, vol. 21, pp. 767–771, Aug 2005.
- [17] L. Acosta, J. J. Rodrigo, J. A. Mendez, G. N. Marichal, and M. Sigut, "Ping-pong player prototype," *IEEE Robotics Automation Magazine*, vol. 10, pp. 44–52, Dec 2003.
- [18] C. H. Lai and T. I. J. Tsay, "Self-learning for a humanoid robotic ping-pong player," *Advanced Robotics*, vol. 25, no. 9-10, pp. 1183–1208, 2011.
- [19] Z. Yu, Y. Liu, Q. Huang, X. Chen, W. Zhang, J. Li, G. Ma, L. Meng, T. Li, and W. Zhang, "Design of a humanoid ping-pong player robot with redundant joints," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 911–916, Dec 2013.
- [20] Y. Sun, R. Xiong, Q. Zhu, J. Wu, and J. Chu, "Balance motion generation for a humanoid robot playing table tennis," in *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 19–25, Oct 2011.
- [21] R. Xiong, Y. Sun, Q. Zhu, J. Wu, and J. Chu, "Impedance control and its effects on a humanoid robot playing table tennis," *International Journal of Advanced Robotic Systems*, vol. 9, no. 5, p. 178, 2012.
- [22] Y. Huang, B. Schölkopf, and J. Peters, "Learning optimal striking points for a ping-pong playing robot," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4587–4592, Sept 2015.
- [23] A. Nakashima, Y. Ogawa, Y. Kobayashi, and Y. Hayakawa, "Modeling of rebound phenomenon of a rigid ball with friction and elastic effects," in *Proceedings of the 2010 American Control Conference*, pp. 1410–1415, June 2010.
- [24] Y. Huang, D. Xu, M. Tan, and H. Su, "Trajectory prediction of spinning ball for ping-pong player robot," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3434–3439, Sept 2011.
- [25] J. Glover and L. P. Kaelbling, "Tracking the spin on a ping pong ball with the quaternion bingham filter," in *IEEE Conference on Robotics and Automation (ICRA)*, 2014.
- [26] Y. Zhang, R. Xiong, Y. Zhao, and J. Wang, "Real-time spin estimation of ping-pong ball using its natural brand," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, pp. 2280–2290, Aug 2015.
- [27] Y. Zhao, R. Xiong, and Y. Zhang, "Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm," *Journal of Intelligent & Robotic Systems*, vol. 87, pp. 407–423, Sep 2017.
- [28] B. Bäuml, O. Birbach, T. Wimböck, U. Frese, A. Dietrich, and G. Hirzinger, "Catching flying balls with a mobile humanoid: System overview and design considerations," in *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 513–520, Oct 2011.
- [29] S. Kim, E. Gribovskaya, and A. Billard, "Learning motion dynamics to catch a moving object," in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pp. 106–111, Dec 2010.

-
- [30] J. Nakanishi, A. Radulescu, D. J. Braun, and S. Vijayakumar, “Spatio-temporal stiffness optimization with switching dynamics,” *Autonomous Robots*, pp. 1–19, 2016.
- [31] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton, NJ, USA: Princeton University Press, 2011.
- [32] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*. Interscience (English translation), 1962.
- [33] A. E. Bryson and Y.-C. Ho, *Applied optimal control : optimization, estimation, and control*. Hemisphere Pub. Corp.; distributed by Halsted Press Washington : New York, rev. printing. ed., 1975.
- [34] H. Schättler and U. Ledzewicz, *Geometric optimal control : theory, methods and examples*. Interdisciplinary applied mathematics, New York, Heidelberg, Dordrecht: Springer, 2012.
- [35] H. W. Sorenson, *Kalman filtering: theory and application*. IEEE Press, Los Alamitos, CA, 1985.
- [36] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. Hoboken (N.J.): John Wiley & Sons, 2006.
- [37] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, “Trajectory planning for optimal robot catching in real-time,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 3719–3726, May 2011.
- [38] S. Schaal, “The SL simulation and real-time control software package,” tech. rep., 2006.
- [39] S. G. Johnson, “The nlopt nonlinear-optimization package.” <http://ab-initio.mit.edu/nlopt>, 2016.
- [40] M. J. D. Powell, *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pp. 51–67. Dordrecht: Springer Netherlands, 1994.
- [41] C. H. Lampert and J. Peters, “Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components,” *J. Real-Time Image Process.*, vol. 7, pp. 31–41, Mar. 2012.
- [42] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.
- [43] D. Bristow, M. Tharayil, and A. Alleyne, “A survey of iterative learning control,” *Control Systems, IEEE*, vol. 26, pp. 96 – 114, june 2006.
- [44] H.-S. Ahn, Y. Q. Chen, and K. Moore, “Iterative learning control: Brief survey and categorization,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, pp. 1099–1121, Nov 2007.
- [45] N. Amann, D. H. Owens, and E. Rogers, “Iterative learning control for discrete time systems with exponential rate of convergence,” 1995.

-
- [46] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems 22 (NIPS 2008)*, Cambridge, MA: MIT Press, pp. 849–856, 2009.
- [47] T. D. Son, G. Pipeleers, and J. Swevers, "Robust monotonic convergent iterative learning control," *IEEE Transactions on Automatic Control*, vol. 61, pp. 1063–1068, April 2016.
- [48] K. E. Avrachenkov, "Iterative learning control based on quasi-newton methods," in *IEEE CDC'98 Proc*, 1998.
- [49] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *Journal of Robotic Systems*, vol. 1, no. 2, pp. 123–140, 1984.
- [50] E. Rogers, K. Galkowski, and D. H. Owens, *Control Systems Theory and Applications for Linear Repetitive Processes*. Springer-Verlag, 2007.
- [51] M. N. of and S. Gunnarsson, "Time and frequency domain convergence properties in iterative learning control," 2002.
- [52] R. W. Longman, "Iterative learning control and repetitive control for engineering practice," *International Journal of Control*, vol. 73, no. 10, pp. 930–954, 2000.
- [53] S. Hillenbrand and M. Pandit, "An iterative learning controller with reduced sampling rate for plants with variations of initial states," *International Journal of Control*, vol. 73, no. 10, pp. 882–889, 2000.
- [54] K.-H. Park and Z. Bien, "A generalized iterative learning controller against initial state error," *International Journal of Control*, vol. 73, no. 10, pp. 871–881, 2000.
- [55] Y. Fang and T. Chow, "2-d analysis for iterative learning controller for discrete-time systems with variable initial conditions," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 50, pp. 722–727, May 2003.
- [56] I. Chin, S. Qin, K. S. Lee, and M. Cho, "A two-stage iterative learning control technique combined with real-time feedback for independent disturbance rejection," *Automatica*, vol. 40, pp. 1913–1922, Nov. 2004.
- [57] J. Z. Kolter and A. Y. Ng, "Policy search via the signed derivative.," in *Robotics: Science and Systems* (J. Trinkle, Y. Matsuoka, and J. A. Castellanos, eds.), The MIT Press, 2009.
- [58] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, pp. 319–340, Apr. 2011.
- [59] J. Ghosh and B. Paden, "Pseudo-inverse based iterative learning control for nonlinear plants with disturbances," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, vol. 5, pp. 5206–5212 vol.5, 1999.
- [60] J. G. P Barnes, "An algorithm for solving non-linear equations based on the secant method," *The Computer Journal*, vol. 8, no. 1, pp. 66–72, 1965.
- [61] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, "Optimization-based iterative learning for precise quadcopter trajectory tracking," *Autonomous Robots*, vol. 33, pp. 103–127, 2012.

-
- [62] J. van den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2074–2081, May 2010.
- [63] G. J. Maeda, I. Manchester, and D. Rye, "In press: Combined ILC and disturbance observer for the rejection of near-repetitive disturbances, with application to excavation," *IEEE Transactions on Control Systems Technology*, March 2015.
- [64] M. Deisenroth and C. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *International Conference on Machine Learning (ICML 2011)*, 2011.
- [65] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. 1989.
- [66] O. Koc, G. Maeda, G. Neumann, and J. Peters, "Optimizing robot striking movement primitives with iterative learning control," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015.
- [67] J. Nonomura, A. Nakashima, and Y. Hayakawa, "Analysis of effects of rebounds and aerodynamics for trajectory of table tennis ball," in *SICE Annual Conference 2010, Proceedings of*, pp. 1567–1572, Aug 2010.
- [68] G. J. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, and J. Peters, "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks," *Autonomous Robots*, vol. 41, no. 3, p. 593–612, 2017.
- [69] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Comput.*, vol. 25, pp. 328–373, Feb. 2013.
- [70] S. M. Khansari-Zadeh and A. Billard, "Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions," *Robotics and Autonomous Systems*, vol. 62, pp. 752–765, 2014.
- [71] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems (NIPS)*, Cambridge, MA: MIT Press., 2013.
- [72] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, pp. 943–957, Oct 2011.
- [73] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.
- [74] G. Obozinski and B. Taskar, "Multi-task feature selection," in *In the workshop of structural Knowledge Transfer for Machine Learning in the 23rd International Conference on Machine Learning (ICML 2006)*. Citeseer, 2006.
- [75] J. Kober, K. Muelling, O. Kroemer, C. Lampert, B. Schoelkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

-
- [76] R. H. Shumway and D. S. Stoffer, “An approach to time series smoothing and forecasting using the em algorithm,” *Journal of Time Series Analysis*, vol. 3, no. 4, pp. 253–264, 1982.
- [77] J. L. Speyer and D. H. Jacobson, *Primer on Optimal Control Theory*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010.

A Appendix

A.1 Parameter Estimation

In order to reach successful performance in real robot table tennis, accurate ball prediction models are needed. For this purpose, we collect data from a human table tennis demonstration recording and estimate the parameters of the ball models. In these sessions, we record the ball position observations from the cameras as well as the robot joint angles. A ball-launcher is used to launch balls with high topspin. The noisy dataset of human table tennis demonstrations

$$\mathcal{D} = \{\mathbf{t}_i, \mathbf{b}_i, \mathbf{q}_i\}_{i=1}^N, \quad (\text{A.1})$$

consists of $N = 90$ trials. Each trial i contains M_i ball position and K_i joint position recordings sorted by time, i.e.,

$$\mathbf{b}_i = \{\mathbf{b}_{ij}\}_{j=1}^{M_i}, \mathbf{q}_i = \{\mathbf{q}_{ij}\}_{j=1}^{K_i}, \quad (\text{A.2})$$

sorted by $\mathbf{t}_i = \{t_{ij}\}_{j=1}^{K_i}$. Typically $K_i > M_i$, e.g., for the Barrett WAM, we record the robot joint values with a frequency of 500 Hz, whereas our vision system outputs ball observations at around 60 Hz.

Whenever the future path of the ball is predicted with the ball models, the accuracy of the predictions using the rebound model (2.10) and the racket contact model (2.14) clearly depend on that of the flight model (2.9). Hence we first start by estimating the parameters of the flight model using nonlinear least squares (NLS). We collect the time stamps and the ball positions detected by the vision system before rebound. The rebound index for each trial i is estimated as

$$\begin{aligned} j_b(i) &= \arg \min_j b_{z_{ij}}, \\ \text{s.t. } & -\frac{w_T}{2} \leq b_{x_{ij}} \leq \frac{w_T}{2}, \\ & y_{\text{edge}} - l_T \leq b_{y_{ij}} \leq y_{\text{edge}}. \end{aligned} \quad (\text{A.3})$$

We then use all of the observations until rebound, $\{(t_{ij}, \mathbf{b}_{ij})_{j=1}^{j_b(i)}\}$, to estimate the parameters g , C_D and C_L . This procedure requires to estimate as well the trial specific topspin parameter, i.e., $\boldsymbol{\omega} = (0, 0, \omega_z)^T$ for some topspin value w_z , separately for each trial $i = 1 \dots, N$.

We use the flight model with the estimated parameters to smoothen the ball path before rebound and after rebound. Using an Extended Kalman Smoother [35], the ball velocities are calculated before and after rebound, $\dot{\mathbf{b}}_{\text{in}}, \dot{\mathbf{b}}_{\text{out}}$ respectively. NLS is again used to estimate the rebound parameters μ_t and ϵ_t . See Figure A.1 for an illustration.

In order to estimate the parameters, i.e., κ and ϵ_r , of the ball-racket interaction model (2.14) we first use the dataset to estimate the striking times T_i . By considering the minimum distance between the ball samples and the demonstrated Cartesian robot trajectories

$$\begin{aligned} j_h(i) &= \arg \min_j \|\mathbf{b}_{ij} - \mathbf{K}_p(\mathbf{q}_{ij})\|_2, \\ T_i &\approx t_{ij_h(i)}, \end{aligned} \quad (\text{A.4})$$

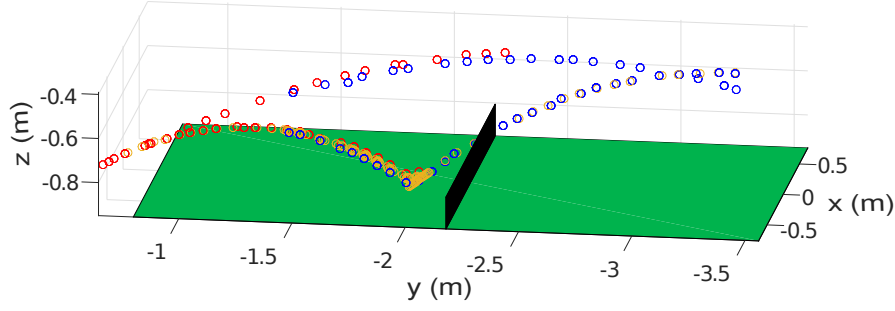


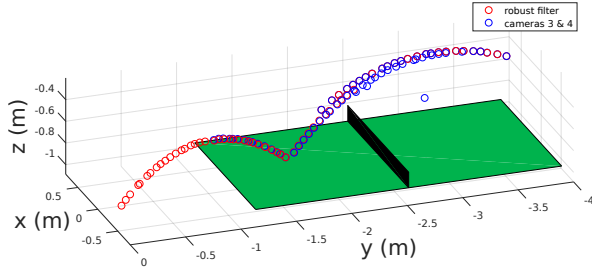
Figure A.1: Using Extended Kalman Smoothing (EKS) to estimate the parameters of the rebound model from actual noisy ball data during a demonstration recording. Ball observations are acquired from two different sets of cameras on opposite sides of the table, shown as red and blue circles respectively. They are then smoothed with the EKS, shown in yellow, to obtain velocity estimates before rebound and just after rebound. Nonlinear least squares is then used to estimate the rebound model parameters μ_t and ϵ_t .

for each trial $i = 1, \dots, N$ we can roughly estimate the striking times T_i . As in estimating the rebound model parameters, the Extended Kalman Smoother is then used to smoothen the ball demonstrations before and after the striking time separately. This procedure results in estimating the incoming and outgoing ball velocities, $\dot{\mathbf{b}}_{\text{in}}(T_i)$, $\dot{\mathbf{b}}_{\text{out}}(T_i)$ as well as the required racket quantities $\mathbf{v}(T_i)$, $\mathbf{n}(T_i)$ at striking times T_i . Linear Least Squares is then used with regularizer $\lambda = 0.001$ to estimate the coefficients κ and ϵ_r . See Table A.1 for the estimated values of all the ball model parameters.

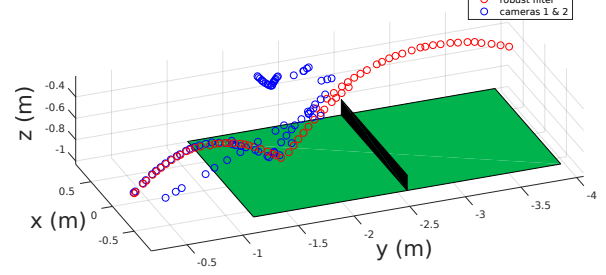
An alternative approach would be to estimate all the model parameters together with a smoothing Expectation-Maximization (EM) [76] algorithm, yielding additionally covariance estimates for noisy ball observations.

Table A.1: Ball model parameter estimates

Parameter	Description	Estimate
C_D	Air drag coefficient	0.141 1/m
C_L	Lift coefficient	0.001 1/rad
g	Gravity	-9.802 m/s ²
μ_t	Coeff. of friction of table	0.102
ϵ_t	Coef. of restitution of table	0.883
κ	Coeff. of friction of racket	0.020
ϵ_r	Coeff. of restitution of racket	0.788



(a) Filtering the noisy and corrupted data acquired from cameras 3 and 4 on the robot side.



(b) Filtering the noisy and corrupted data acquired from cameras 1 and 2 opposite to the robot side.

Figure A.2: Kalman Filtering with simultaneous outlier detection. The ball detection algorithm sometimes outputs outliers, typically more as the ball approaches the racket. Such corrupted data can be identified and discarded using the covariance of the Kalman Filter.

A.2 Robust Kalman Filtering

The ball detection algorithm sometimes outputs outliers, possibly meters away from the actual ball. This typically happens more as the ball approaches the racket and new ball observations become more valuable. In order to prevent the outliers from ruining the estimation and the overall performance, we have implemented a robust EKF that does not perform measurement updates, if the ball observations lie more than 2 standard deviations away from the predicted state. See Figure A.2 for actual table tennis ball data. We adjust the covariance estimates $\Sigma(t)$ accordingly to make this procedure work in practice, e.g., covariances are initialized with a large Σ_{init} value and the noise covariances $\mathbf{W}(t) \approx 10^{-3}\mathbf{I}$ are adjusted to make sure that $\Sigma(t)$ decreases suitably over time.

A.3 Derivations for Chapter 2

We will give a self-contained derivation of the fact that the solutions to the parametric optimization problem (2.55) are also locally optimal solutions to (2.1). The same holds for the solutions of (2.30) under the additional racket constraints (2.18) – (2.20). First, we start by deriving the fixed final-time version of the Minimum Principle (MP) from the Hamilton Jacobi Bellman equation (HJB). Sufficient continuity of the introduced variables are assumed throughout to simplify the presentation. The solutions of the following fixed final-time optimal control problem

$$\begin{aligned} \min_{\mathbf{u} \in \mathcal{U}} \int_0^T l(\mathbf{x}, \mathbf{u}) \, dt + \phi(\mathbf{x}(T)), \\ \text{s.t. } \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \end{aligned} \tag{A.5}$$

from arbitrary initial conditions are given by the solutions of the HJB partial differential equation (PDE)

$$\min_{\pi(\mathbf{x}, t)} \left(l(\mathbf{x}, \pi) + \frac{\partial V(\mathbf{x}, t)}{\partial \mathbf{x}}^T \mathbf{f}(\mathbf{x}, \pi(\mathbf{x}, t)) \right) + \frac{\partial V(\mathbf{x}, t)}{\partial t} = 0, \quad (\text{A.6})$$

$$V(\mathbf{x}, T) = \phi(\mathbf{x}(T)). \quad (\text{A.7})$$

For every state \mathbf{x} , $V(\mathbf{x}, t)$ is the optimal cost to (A.5) for following the optimal policy $\pi(\mathbf{x}, t)$, also called the *value function*. MP [32] can be seen in the smooth case as a solution technique¹ that reduces (A.6) to a family of ordinary differential equations (ODE). Introducing the *momenta* $\mathbf{p}(t) = \frac{\partial V}{\partial \mathbf{x}}$ and the *Hamiltonian* $\mathcal{H}(\mathbf{x}, \mathbf{u}, \mathbf{p}) = l(\mathbf{x}, \mathbf{u}) + \mathbf{p}(t)^T \mathbf{f}(\mathbf{x}, \mathbf{u})$, the HJB equation (A.6) becomes

$$\min_{\mathbf{u}(t)} \mathcal{H}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}(t)) + \frac{\partial V}{\partial t} = 0. \quad (\text{A.8})$$

Given an initial condition $\mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^n$, solving (A.8) is reduced to solving a Boundary Value Problem (BVP)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}^*(t)), \quad (\text{A.9})$$

$$\dot{\mathbf{p}}(t) = \frac{\partial}{\partial t} \frac{\partial V(\mathbf{x}, t)}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \frac{\partial V}{\partial t} = - \frac{\partial \mathcal{H}(\mathbf{x}, \mathbf{u}^*, \mathbf{p})}{\partial \mathbf{x}}, \quad (\text{A.10})$$

$$\mathbf{u}^*(t) = \arg \min_{\mathbf{u}} \mathcal{H}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}(t)), \quad (\text{A.11})$$

with prescribed boundary values

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{x}_0, \\ \mathbf{p}(T) &= \frac{\partial V(\mathbf{x}, T)}{\partial \mathbf{x}} \Big|_{\mathbf{x}(T)} = \frac{\partial \phi(\mathbf{x})}{\partial \mathbf{x}} \Big|_{\mathbf{x}(T)}. \end{aligned} \quad (\text{A.12})$$

The state equation (A.9) and the *costate* equation (A.10) that describes the evolution of the introduced momenta are $2n$ first-order coupled ODEs that can be solved numerically with a BVP solver.

Free final-time

If the final time T is free, the optimal control problem

$$\begin{aligned} \min_{\mathbf{u}, T} \quad & \int_0^T l(\mathbf{x}, \mathbf{u}) \, dt + \phi(\mathbf{x}(T), T), \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \end{aligned} \quad (\text{A.13})$$

¹ More generally, the solution technique is called the *method of characteristics* for hyperbolic PDEs.

can be transformed to the standard form (A.5) by using the transformation $\tau = T t$

$$\begin{aligned} \min_{\mathbf{u}, T} \int_0^1 T l(\mathbf{x}, \mathbf{u}) + \phi(\mathbf{x}(T), T) d\tau, \\ \text{s.t. } \frac{d\mathbf{x}}{d\tau} = T \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau)), \end{aligned} \quad (\text{A.14})$$

and the associated HJB, similar to (A.8) can be written as

$$\min_{\mathbf{u}(t), T} (T \mathcal{H}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}(t)) + \phi(\mathbf{x}(T), T)) + \frac{\partial V}{\partial t} = 0. \quad (\text{A.15})$$

In this case, the state and the costate differential equations are preserved

$$\begin{aligned} \frac{d\mathbf{x}}{d\tau} &= \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau)), \\ \frac{d\mathbf{p}}{d\tau} &= -\frac{\partial \mathcal{H}(\mathbf{x}, \mathbf{u}^*, \mathbf{p})}{\partial \mathbf{x}}, \end{aligned} \quad (\text{A.16})$$

while the additional minimization with respect to T in (A.15) yields an additional *time variation* to constrain the Hamiltonian

$$\mathcal{H}(\mathbf{x}, \mathbf{u}^*, \mathbf{p}) + \frac{\partial \phi(\mathbf{x}, T)}{\partial T} = 0. \quad (\text{A.17})$$

The $2n$ first-order coupled ODEs (A.16) can again be solved numerically with a BVP solver, but with the additional parameterization T of the boundary values (A.12) that is resolved implicitly through (A.17).

Equality constraints

When m equality constraints at final time T are present rather than a final cost term ϕ , the (generalized) HJB equation (A.6) still holds but the boundary term (A.7) constraining the value function is replaced with an equality constraint $\Psi(\mathbf{x}(T), T) = \mathbf{0}$. The value function is undefined for states violating the equality constraint at final time, and zero otherwise [77].

Equality constraints at the boundaries do not affect the differential equations (A.16), they are still valid for $t \in (0, T)$. The time variation (A.17) has to be adapted with the addition of Lagrange multipliers $\boldsymbol{\nu} \in \mathbb{R}^m$

$$\mathcal{H}(\mathbf{x}, \mathbf{u}^*, \mathbf{p}) + \frac{\partial \Psi(\mathbf{x}(T), T)^T \boldsymbol{\nu}}{\partial T} = 0. \quad (\text{A.18})$$

This can be combined with the boundary term

$$\mathbf{p}(T) = \left. \frac{\partial \Psi(\mathbf{x}(T), T)^T \boldsymbol{\nu}}{\partial \mathbf{x}} \right|_{\mathbf{x}(T)}, \quad (\text{A.19})$$

to form the *transversality conditions*²

$$\begin{bmatrix} -\mathcal{H}(T) \\ \mathbf{p}(T) \end{bmatrix} = D\Psi^T \boldsymbol{\nu} = \begin{bmatrix} \frac{\partial \Psi(\mathbf{x}(T), T)}{\partial T} & \frac{\partial \Psi(\mathbf{x}(T), T)}{\partial \mathbf{x}} \end{bmatrix}^T \boldsymbol{\nu}. \quad (\text{A.20})$$

When solving (A.16) starting from \mathbf{x}_0 , the additional parameters $\boldsymbol{\nu} \in \mathbb{R}^m$ and T can be found by solving the nonlinear set of equations (A.20) along with $\Psi(\mathbf{x}(T), T) = \mathbf{0}$.

Inequality constraints

When m inequality constraints $\Phi(\mathbf{x}(T), T) \leq \mathbf{0}$ are present at final time T rather than equalities, the equality constraints are replaced with the *Karush-Kuhn-Tucker (KKT) conditions*,

$$\begin{aligned} \Phi(\mathbf{x}(T), T) &\leq \mathbf{0}, \\ \Phi^T(\mathbf{x}(T), T) \boldsymbol{\nu} &= \mathbf{0}, \\ \boldsymbol{\nu} &\geq \mathbf{0}. \end{aligned} \quad (\text{A.21})$$

These are known as primal feasibility, complementary slackness and dual feasibility conditions, respectively. Transversality conditions (A.20) still hold.

Local & Global Sufficiency

The state and costate equations (A.16) along with the transversality conditions (A.20) are only *necessary* conditions [32]. Necessary solutions can be strengthened with HJB to form sufficient conditions for global optimality. Solutions found with MP are (globally) sufficient if the optimal controls (A.11) satisfy HJB (A.6) for a continuously differentiable $V(\mathbf{x}, t) \in \mathcal{C}^1(\mathbb{R}^n, [0, T])$, which is hard to find for free-time control problems. Sufficient conditions for local optimality, on the other hand, are easier to verify, see for example, Chapter 6 of [33] or Chapter 5 of [77].

Numerical solution by direct optimization

Numerical integration of BVP when free-parameters are present can be very difficult and/or time consuming. An alternative approach suitable for generating minimum-acceleration trajectories is given in Chapter 3. Inserting (A.9) – (A.11) into (2.1), yields for our particular problem Equation (2.25). The integrand of the cost functional, parameterized by the free parameters $\mathbf{a}_3, \mathbf{a}_2, T$ (or equivalently, $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$ using Equation (2.8)) of the momenta, is integrated from 0 to T to form the cost function (2.30) of the Focused Player optimization. The Defensive Player optimization (2.55) changes the racket equality constraints to ball landing inequalities while adding additional penalties $\phi_{\text{pen}}(\mathbf{q}_f, \dot{\mathbf{q}}_f, T)$ to the cost function.

Local optima of the augmented cost functionals (2.28) and (2.54) can be shown to satisfy the remaining necessary conditions for optimality, namely the transversality conditions (A.20) and (A.21). Parameterizing the Hamiltonian \mathcal{H} and the momenta \mathbf{p} at striking time w.r.t. optimization variables $\mathbf{q}_f, \dot{\mathbf{q}}_f, T$, (A.20) and (A.21) are consequences of the necessary first order optimality conditions for Problems (2.30) and (2.55), respectively.

² With slight abuse of notation, the derivatives of the boundary term Ψ are evaluated at *optimized* T and $\mathbf{x}(T)$, which is found by evolving the state equation till T .

A.4 Derivations for Chapter 3

We provide in this section self-contained derivations of the cautious ILC update rule, given in Equations (3.24) and simplified in (3.29). Consider the following optimal control problem

$$\min_{\delta \mathbf{u}} \sum_{j=1}^N \mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [\mathbf{e}_{k+1,j}^T \mathbf{Q}_j \mathbf{e}_{k+1,j} + \delta \mathbf{u}_{k+1,j}^T \mathbf{R}_j \delta \mathbf{u}_{k+1,j}], \quad (\text{A.22})$$

$$\text{s.t. } \mathbf{e}_{k+1,j+1} = \mathbf{A}_j \mathbf{e}_{k+1,j} + \mathbf{B}_j \mathbf{u}_{k+1,j} + \mathbf{d}_{j+1}, \quad (\text{A.23})$$

where the linear time-varying system matrices $\mathbf{A}_j, \mathbf{B}_j$ are random variables with known means and variances. Since $\mathbf{u}_{k+1,j} = \mathbf{u}_{k,j} + \delta \mathbf{u}_{k,j}$, we can rewrite (A.23) as

$$\mathbf{e}_{k+1,j+1} = \mathbf{A}_j \mathbf{e}_{k+1,j} + \mathbf{B}_j \delta \mathbf{u}_{k,j} + \bar{\mathbf{d}}_{j+1}, \quad (\text{A.24})$$

$$\bar{\mathbf{d}}_{j+1} = \mathbf{B}_j \mathbf{u}_{k,j} + \mathbf{d}_{j+1}. \quad (\text{A.25})$$

The iteration index k will be removed until the last subsection for the convenience of the reader. Notice that the Value Function for the optimal control problem (A.23) is a quadratic function of the errors along the trajectory,

$$V(\mathbf{e}, j) = \mathbf{e}^T \mathbf{P}_j \mathbf{e} + 2\mathbf{e}^T \mathbf{b}_j + c_j, \quad (\text{A.26})$$

for time-varying matrices $\mathbf{P}_j \in \mathbb{R}^{2n \times 2n}$, vectors $\mathbf{b}_j \in \mathbb{R}^{2n}$ and $c_j \in \mathbb{R}$. We can then apply dynamic programming to compute the optimal solution recursively

$$\begin{aligned} V(\mathbf{e}_j, j) &= \min_{\delta \mathbf{u}_j} (\mathbf{e}_j^T \mathbf{Q}_j \mathbf{e}_j + \delta \mathbf{u}_j^T \mathbf{R}_j \delta \mathbf{u}_j + V(\mathbf{e}_{j+1}, j+1)), \\ V(\mathbf{e}_{j+1}, j+1) &= \mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [2\mathbf{b}_{j+1}^T (\mathbf{A}_j \mathbf{e}_j + \mathbf{B}_j \delta \mathbf{u}_j + \bar{\mathbf{d}}_{j+1}) + c_{j+1} \\ &\quad + (\mathbf{A}_j \mathbf{e}_j + \mathbf{B}_j \delta \mathbf{u}_j + \bar{\mathbf{d}}_{j+1})^T \mathbf{P}_{j+1} (\mathbf{A}_j \mathbf{e}_j + \mathbf{B}_j \delta \mathbf{u}_j + \bar{\mathbf{d}}_{j+1})]. \end{aligned} \quad (\text{A.27})$$

The recursion starts from $\mathbf{P}_N = \mathbf{Q}_N$. Taking derivative w.r.t. $\delta \mathbf{u}_j$ of the right-hand side, we get

$$\mathbf{R}_j \delta \mathbf{u}_j + (\mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [\mathbf{B}_j^T \mathbf{P}_{j+1} \mathbf{A}_j] \mathbf{e}_j + \mathbb{E}_{\mathbf{B}_j} [\mathbf{B}_j^T \mathbf{P}_{j+1} \mathbf{B}_j] \delta \mathbf{u}_j + \mathbb{E}_{\mathbf{B}_j} [\mathbf{B}_j^T (\mathbf{P}_{j+1} \bar{\mathbf{d}}_{j+1} + \mathbf{b}_{j+1})]) = 0. \quad (\text{A.28})$$

Solving (A.28) for the optimal control input compensations, and arranging using the notation in (3.24)

$$\delta \mathbf{u}_j = \mathbf{K}_j \mathbf{e}_j - \Phi_j^{-1} \ell_j, \quad (\text{A.29})$$

$$\mathbf{K}_j = -\Phi_j^{-1} \Psi_j, \quad (\text{A.30})$$

$$\Phi_j = \mathbf{R}_j + \mathbb{E}_{\mathbf{B}_j} [\mathbf{B}_j^T \mathbf{P}_{j+1} \mathbf{B}_j], \quad (\text{A.31})$$

$$\Psi_j = \mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [\mathbf{B}_j^T \mathbf{P}_{j+1} \mathbf{A}_j], \quad (\text{A.32})$$

$$\ell_j = \mathbb{E}_{\mathbf{B}_j} [\mathbf{B}_j^T (\mathbf{P}_{j+1} \bar{\mathbf{d}}_{j+1} + \mathbf{b}_{j+1})]. \quad (\text{A.33})$$

In order to derive a Riccati-like equation, we plug (A.4) into (A.27), and using (A.26) get

$$\begin{aligned} \mathbf{e}^T \mathbf{P}_j \mathbf{e} + 2\mathbf{e}^T \mathbf{b}_j + c_j &= \mathbf{e}_j^T \mathbf{Q}_j \mathbf{e}_j + \mathbf{e}_j^T (\Psi_j^T \Phi_j^{-1} \mathbf{R}_j \Phi_j^{-1} \Psi_j) \mathbf{e}_j + 2\ell_j^T \Phi_j^{-1} \mathbf{R}_j \Phi_j^{-1} \Psi_j \mathbf{e}_j + \ell_j^T \Phi_j^{-1} \mathbf{R}_j \Phi_j^{-1} \ell_j \\ &\quad + \mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [(\bar{\mathbf{A}}_j \mathbf{e}_j + \mathbf{m}_j)^T \mathbf{P}_{j+1} (\bar{\mathbf{A}}_j \mathbf{e}_j + \mathbf{m}_j)] + 2\mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [(\bar{\mathbf{A}}_j \mathbf{e}_j + \mathbf{m}_j)^T \mathbf{b}_{j+1}] + c_{j+1}, \end{aligned} \quad (\text{A.34})$$

where we have introduced the terms

$$\bar{\mathbf{A}}_j = \mathbf{A}_j + \mathbf{B}_j \mathbf{K}_j, \quad (\text{A.35})$$

$$\mathbf{m}_j = \bar{\mathbf{d}}_{j+1} - \mathbf{B}_j \Phi_j^{-1} \ell_j. \quad (\text{A.36})$$

Checking for the equality of the quadratic terms we get, after some cancellations,

$$\mathbf{P}_j = \mathbf{Q}_j + \mathbf{M}_j - \Psi_j^T \Phi_j^{-1} \Psi_j, \quad (\text{A.37})$$

$$\mathbf{M}_j = \mathbb{E}_{\mathbf{A}_j} [\mathbf{A}_j^T \mathbf{P}_{j+1} \mathbf{A}_j], \quad (\text{A.38})$$

$$\mathbf{b}_j = \Psi_j^T \Phi_j^{-1} \mathbf{R}_j \Phi_j^{-1} \ell_j + \mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [\bar{\mathbf{A}}_j^T (\mathbf{P}_{j+1} \mathbf{m}_j + \mathbf{b}_{j+1})]. \quad (\text{A.39})$$

Rewriting the feedforward recursion

The control input compensations calculated in (A.4) can be simplified significantly by noting that the last three terms in the feedforward recursion of (A.4)

$$\mathbf{b}_j = \mathbb{E}[\bar{\mathbf{A}}_j^T (\mathbf{b}_{j+1} + \mathbf{P}_{j+1} \bar{\mathbf{d}}_{j+1})] - \mathbb{E}[\mathbf{A}_j^T \mathbf{P}_{j+1} \mathbf{B}_j] \Phi_j^{-1} \ell_j - \mathbf{K}_j^T \mathbb{E}[\mathbf{B}_j^T \mathbf{P}_{j+1} \mathbf{B}_j] \Phi_j^{-1} \ell_j - \mathbf{K}_j^T \mathbf{R}_j \Phi_j^{-1} \ell_j, \quad (\text{A.40})$$

cancel out, leaving

$$\mathbf{b}_j = \mathbb{E}_{\mathbf{A}_j, \mathbf{B}_j} [\bar{\mathbf{A}}_j^T (\mathbf{b}_{j+1} + \mathbf{P}_{j+1} \bar{\mathbf{d}}_{j+1})]. \quad (\text{A.41})$$

The cancellations can be seen easily by rewriting the first term in terms of the feedback matrix and grouping the last two terms together

$$-\mathbf{K}_j^T \Phi_j \Phi_j^{-1} \ell_j + \mathbf{K}_j^T \ell_j = 0. \quad (\text{A.42})$$

Simplifying the feedforward recursion

The feedforward recursion in (A.41) still requires the explicit estimation of disturbances. This equation can be simplified further by rewriting the disturbances in terms of the previous trial errors

$$\begin{aligned} \bar{\mathbf{d}}_{j+1} &= \mathbf{e}_{k,j+1} - \mathbf{A}_j \mathbf{e}_{k,j}, \\ \ell_j &= \mathbb{E}[\mathbf{B}_j^T (\mathbf{P}_{j+1} \mathbf{e}_{k,j+1} + \mathbf{b}_{j+1})] - \Psi_j \mathbf{e}_{k,j}. \end{aligned} \quad (\text{A.43})$$

Introducing $\boldsymbol{\nu}_{j+1} = \mathbf{P}_{j+1} \mathbf{e}_{k,j+1} + \mathbf{b}_{j+1}$, we can rewrite the optimal control input compensations as

$$\delta \mathbf{u}_j = \mathbf{K}_j (\mathbf{e}_{k+1,j} - \mathbf{e}_{k,j}) - \Phi_j^{-1} \mathbb{E}[\mathbf{B}_j^T \boldsymbol{\nu}_{j+1}]. \quad (\text{A.44})$$

Rewriting (A.41) in terms of $\boldsymbol{\nu}_j$, we get

$$\boldsymbol{\nu}_j = \mathbb{E}[\bar{\mathbf{A}}_j^T \boldsymbol{\nu}_{j+1}] + (\mathbf{P}_j - \mathbb{E}[(\mathbf{A}_j + \mathbf{B}_j \mathbf{K}_j)^T \mathbf{P}_{j+1} \mathbf{A}_j]) \mathbf{e}_{k,j}, \quad (\text{A.45})$$

since $\mathbf{P}_j = \mathbf{Q}_j + \mathbf{M}_j - \Psi_j^T \Phi_j^{-1} \Psi_j$, the last term becomes

$$(\mathbf{P}_j - \mathbf{M}_j - \mathbf{K}_j^T \Psi_j) \mathbf{e}_{k,j} = \mathbf{Q}_j \mathbf{e}_{k,j}, \quad (\text{A.46})$$

hence, the feedforward recursion defining (A.44) can be computed independently of disturbance estimates

$$\boldsymbol{\nu}_j = \mathbb{E}[\bar{\mathbf{A}}_j^T \boldsymbol{\nu}_{j+1}] + \mathbf{Q}_j \mathbf{e}_{k,j}, \quad j = 1, \dots, N-1, \quad (\text{A.47})$$

starting from $\boldsymbol{\nu}_N = \mathbf{0}$.

Publications & CV

Publications

The work presented in this thesis contributed to the following publications:

Journal Papers

1. **Koç, O.**, Maeda, G., and Peters, J. "Online optimal trajectory generation for robot table tennis." *Robotics and Autonomous Systems (RAS)*, 2018, 10.1016/j.robot.2018.03.012
2. **Koç, O.**, Maeda, G., and Peters, J. "Optimizing Execution of Dynamic Goal-Directed Robot Movements with Learning Control" *IEEE Transactions on Robotics*. (accepted)
3. **Koç, O.**, and Peters, J. "Learning to serve: an experimental study for a new learning from demonstrations framework" *IEEE Robotics and Automation Letters (ICRA/RA-L)*, 2019, 10.1109/LRA.2019.2896466

Conference Papers

1. Maeda, G., **Koç, O.**, and Morimoto, J. "Reinforcement Learning of Phase Oscillators for Adaptation to Fast Moving Targets: Application to a Humanoid Basketball Player." *2nd Conference on Robot Learning (CoRL)*, 2018.
2. Huang, Y., Büchler, D., **Koç, O.**, Schölkopf, B., Peters, J. "Jointly Learning Trajectory Generation and Hitting Point Prediction in Robot Table Tennis." *16th IEEE-RAS International Conference on Humanoid Robots*, 2016.
3. **Koç, O.**, Maeda, G., and Peters, J. "A New Trajectory Generation Framework in Robotic Table Tennis." *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2016.
4. **Koç, O.**, Maeda, G., Neumann, G. and Peters, J. " Optimizing Robot Striking Movement Primitives with Iterative Learning Control." *15th IEEE-RAS International Conference on Humanoid Robots*, 2015.

Curriculum Vitae - Okan Koç

Contact Information

Quenstedtstrasse 32, 72076
Tuebingen, Germany

+49 172-685-6127
okan.koc@tuebingen.mpg.de

Research Interests

Optimal Control, Learning Control, Movement Primitives, Reinforcement Learning.

Education

Max Planck Institute for Intelligent Systems, Tübingen, Germany

Ph.D. candidate, Robotics, *Thesis Successfully Defended*: October 2018

Thesis Topic: *Optimal Trajectory Generation and Learning Control in Robot Table Tennis*

Advisor: Prof. Dr. Jan Peters

ETH Zürich, Zürich, Switzerland

M.S., Applied Mathematics, 2013

Topic: *Gaussian Process Optimization Based Learning for Trajectory Tracking*

Advisors: Prof. Dr. Raffaello D'Andrea, Prof. Dr. Andreas Krause, Prof. Dr. Hans Rudolf Künsch.

GPA: 5.35/6

Bogazici University, Istanbul, Turkey

B.S., Electrical & Electronics Engineering and Mathematics (Double Major), 2009

GPA: 3.16/4

A.5 Papers Reviewed

IROS 2015, RSS 2017, IJRR (2017), HRI 2017, IROS 2018, CORL 2018.

Work Experience

Applied Scientist at Amazon, Tuebingen.

since March 2019

Engineering consultant at Rapyuta Robotics, Japan.

November 2018 - February 2019

Intern at ATR, Japan.

September 2017 - November 2017

Trajectory optimization with a humanoid robot.

R&D specialist at Inveon, Turkey.

March 2014 - June 2014

Recommendation engine research, search platform development.

Assistant at ETH Zürich.	May 2013 - July 2013
<i>Implementation of an Object Search algorithm using Gaussian Mixtures.</i>	
Intern at Sony Corporation, Tokyo, Japan.	June 2012 - September 2012
<i>Clustering, outlier detection on Sony TV log files.</i>	
Intern at ABB Corporate Research, Switzerland.	August 2011 - January 2012
<i>Online stochastic optimization for photovoltaics.</i>	
<i>Maximum Power Point Tracking (MPPT) using restless bandits.</i>	
Intern at Inveon, Turkey.	April - July 2010
<i>Test engineer (.NET web applications, web pages), data analyst (MS SQL).</i>	
Junior consultant at ValueTeam, Turkey.	September 2009 - February 2010
<i>Oracle Siebel CRM (Customer Relationship Management software)</i>	
<i>installation at AVEA, a major Turkish telecommunications company.</i>	

References

Jan Peters
Professor for Machine Learning in Robotics,
FB-Informatik, FG-IAS,
TU Darmstadt.
Phone: +49-6151-16-25373
E-mail: mail@jan-peters.net

Bernhard Schölkopf
Director,
Department of Empirical Inference,
Max Planck Institute for Intelligent Systems.
Phone: +49-7071-601-553
Email: sekretariat-schoelkopf@tuebingen.mpg.de

Skills

Computer Programming:
C, C++, Java, Python, UNIX shell scripting, GNU make, SQL, MATLAB.

Languages:
Turkish (native),
English (fluent, TOEFL 115/120, GRE Verbal 570/800),
German (competent, Goethe Institute Level C1, 11/14),
Spanish (basic).

Interests

Playing guitar and singing, philosophy, literature, traveling.